**Data Structures 2023-2**

**Lab 02**

**ID: 2021136150 Name: 양 동 재**

1. **Task-1: Matrix Data Type**

Develop a Matrix Data Type that has the following functionalities

\_\_init\_\_(row, col) : it initializes a matrix of size rows x cols with random values

\_\_str\_\_() : It returns the string representation of matrix

\_\_repr\_\_() : It is representation of Matrix object

\_\_add\_\_() : it multiplies two matrices and returns a resultant matrix

\_\_sub\_\_() : it subtracts two matrices and returns a resultant matrix

\_\_mul\_\_() : it multiplies two matrices and returns a resultant matrix

transpose() :It returns the transpose of the matrix

**Code**

**class** **Matrix**:

**def** \_\_init\_\_(self, rows, cols, f):

self.M = []

**if** f == 'r':

self.rMatrix(rows, cols)

**elif** f == 'z':

self.zMatrix(rows, cols)

**def** rMatrix(self, rows, cols):

**for** \_ **in** range(rows):

self.M.append([random.randint(1, 10) **for** \_ **in** range(cols)])

**def** zMatrix(self, rows, cols):

**for** \_ **in** range(rows):

self.M.append([0] \* cols)

**def** mPrint(self):

**for** row **in** self.M:

print(row)

**def** \_\_str\_\_(self):

**return** "**\n**".join([" ".join(map(str, row)) **for** row **in** self.M])

**def** \_\_repr\_\_(self):

**return** str(self)

**def** \_\_add\_\_(self, other):

**if** len(self.M) != len(other.M) **or** len(self.M[0]) != len(other.M[0]):

**raise** **ValueError**("Matrix dimensions do not match for addition")

result = Matrix(len(self.M), len(self.M[0]), 'z')

**for** i **in** range(len(self.M)):

**for** j **in** range(len(self.M[0])):

result.M[i][j] = self.M[i][j] + other.M[i][j]

**return** result

**def** \_\_sub\_\_(self, other):

**if** len(self.M) != len(other.M) **or** len(self.M[0]) != len(other.M[0]):

**raise** **ValueError**("Matrix dimensions do not match for subtraction")

result = Matrix(len(self.M), len(self.M[0]), 'z')

**for** i **in** range(len(self.M)):

**for** j **in** range(len(self.M[0])):

result.M[i][j] = self.M[i][j] - other.M[i][j]

**return** result

**def** \_\_mul\_\_(self, other):

**if** len(self.M[0]) != len(other.M):

**raise** **ValueError**("Matrix dimensions are not compatible for multiplication")

result = Matrix(len(self.M), len(other.M[0]), 'z')

**for** i **in** range(len(self.M)):

**for** j **in** range(len(other.M[0])):

result.M[i][j] = sum(self.M[i][k] \* other.M[k][j] **for** k **in** range(len(self.M[0])))

**return** result

**def** transpose(self):

rows, cols = len(self.M), len(self.M[0])

result = Matrix(cols, rows, 'z')

**for** i **in** range(rows):

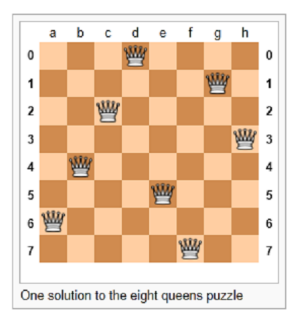
**for** j **in** range(cols):

result.M[j][i] = self.M[i][j]

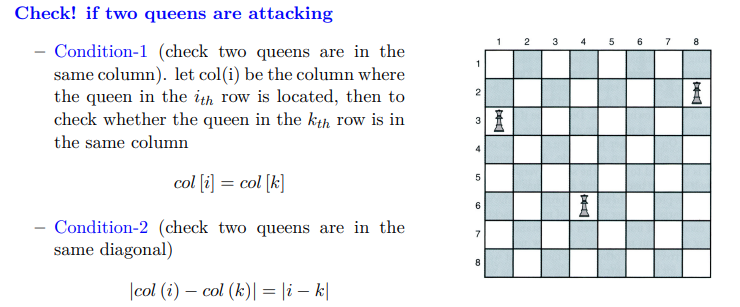
**return** result

**Results/Output**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | | **Result** | |
| **Case1** | **Case2** |
| **Matrix(5,5,’r’)** | **Matrix(5,5,’r’)** |
| **Matrix Initialization** | **Matrix1** | **[5, 3, 4, 8, 7]**  **[6, 3, 10, 10, 5]**  **[9, 6, 1, 7, 9]**  **[7, 1, 4, 1, 5]**  **[1, 9, 6, 7, 8]** | **[3, 4, 4, 3, 4]**  **[1, 4, 9, 8, 3]**  **[4, 1, 6, 3, 8]**  **[2, 9, 10, 8, 2]**  **[2, 8, 4, 9, 6]** |
| **Matrix(5,5,’z’)** | **Matrix(5,5,’r’)** |
| **Matrix2** | **[0, 0, 0, 0, 0]**  **[0, 0, 0, 0, 0]**  **[0, 0, 0, 0, 0]**  **[0, 0, 0, 0, 0]**  **[0, 0, 0, 0, 0]** | **[7, 10, 2, 5, 2]**  **[10, 2, 7, 5, 2]**  **[6, 4, 8, 9, 4]**  **[6, 8, 5, 9, 9]**  **[1, 4, 7, 1, 10]** |
| **Matrix Addition** | | **[5, 3, 4, 8, 7]**  **[6, 3, 10, 10, 5]**  **[9, 6, 1, 7, 9]**  **[7, 1, 4, 1, 5]**  **[1, 9, 6, 7, 8]** | **[10, 14, 6, 8, 6]**  **[11, 6, 16, 13, 5]**  **[10, 5, 14, 12, 12]**  **[8, 17, 15, 17, 11]**  **[3, 12, 11, 10, 16]** |
| **Matrix Subtraction** | | **[-5, -3, -4, -8, -7]**  **[-6, -3, -10, -10, -5]**  **[-9, -6, -1, -7, -9]**  **[-7, -1, -4, -1, -5]**  **[-1, -9, -6, -7, -8]** | **[4, 6, -2, 2, -2]**  **[9, -2, -2, -3, -1]**  **[2, 3, 2, 6, -4]**  **[4, -1, -5, 1, 7]**  **[-1, -4, 3, -8, 4]** |
| **Matrix Multiplication** | | **[0, 0, 0, 0, 0]**  **[0, 0, 0, 0, 0]**  **[0, 0, 0, 0, 0]**  **[0, 0, 0, 0, 0]**  **[0, 0, 0, 0, 0]** | **[107, 94, 109, 102, 97]**  **[152, 130, 163, 181, 148]**  **[100, 122, 134, 114, 141]**  **[214, 150, 201, 219, 154]**  **[178, 148, 179, 173, 177]** |
| **Matrix Transpose** | | **[5, 6, 9, 7, 1]**  **[3, 3, 6, 1, 9]**  **[4, 10, 1, 4, 6]**  **[8, 10, 7, 1, 7]**  **[7, 5, 9, 5, 8]** | **[3, 1, 4, 2, 2]**  **[4, 4, 1, 9, 8]**  **[4, 9, 6, 10, 4]**  **[3, 8, 3, 8, 9]**  **[4, 3, 8, 2, 6]** |

1. **Task-2: EightQueens Data Type**

* The eight-queens puzzle is the problem of placing eight chess queens on an 8x8 chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.
* A basic iterative algorithm starts by initially place the eight queens at random on the board subject to the constraint that there is only one queen on each row and column

****

**Write a class/Type/Data Type to solve this puzzle**

**Code**

**class** **EightQueens**:

rnb = random.Random()

**def** \_\_init\_\_(self):

self.board = list(range(8))

**def** runEQ(self, nos):

found = 0

tries = 0

**while** found < nos:

EightQueens.rnb.shuffle(self.board)

tries += 1

**if** **not** self.has\_clash():

found += 1

print("Solution: **{}**, **{}**".format(found, self.board))

tries = 0

**def** has\_clash(self):

**for** col **in** range(1, len(self.board)):

**if** self.col\_clashes(col):

**return** **True**

**return** **False**

**def** col\_clashes(self, col):

**for** i **in** range(col):

**if** self.dclashes(i, self.board[i], col, self.board[col]):

**return** **True**

**return** **False**

**def** dclashes(self, x0, y0, x1, y1):

d1 = abs(x0 - x1)

d2 = abs(y0 - y1)

**return** d1 == d2

**Results/Output**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Result** | **Visualized solution 1** | | | | | | | |
| **Solution: 1, [3, 0, 4, 7, 1, 6, 2, 5]**  **Solution: 2, [1, 5, 0, 6, 3, 7, 2, 4]**  **Solution: 3, [5, 2, 6, 3, 0, 7, 1, 4]**  **Solution: 4, [2, 6, 1, 7, 4, 0, 3, 5]**  **Solution: 5, [4, 1, 3, 5, 7, 2, 0, 6]**  **Solution: 6, [5, 2, 0, 7, 3, 1, 6, 4]**  **Solution: 7, [2, 5, 7, 1, 3, 0, 6, 4]**  **Solution: 8, [3, 6, 0, 7, 4, 1, 5, 2]**  **Solution: 9, [5, 2, 0, 6, 4, 7, 1, 3]**  **Solution: 10, [3, 6, 2, 7, 1, 4, 0, 5]**  **Solution: 11, [5, 0, 4, 1, 7, 2, 6, 3]**  **Solution: 12, [3, 0, 4, 7, 1, 6, 2, 5]**  **Solution: 13, [4, 2, 0, 5, 7, 1, 3, 6]**  **Solution: 14, [4, 6, 1, 5, 2, 0, 7, 3]**  **Solution: 15, [6, 3, 1, 4, 7, 0, 2, 5]** |  | **O** |  |  |  |  |  |  |
|  |  |  |  | **O** |  |  |  |
|  |  |  |  |  |  | **O** |  |
| **O** |  |  |  |  |  |  |  |
|  |  | **O** |  |  |  |  |  |
|  |  |  |  |  |  |  | **O** |
|  |  |  |  |  | **O** |  |  |
|  |  |  | **O** |  |  |  |  |
| **A screenshot of a computer  Description automatically generated** | | | | | | | | |

1. **Task-3: TicTacToe Data Type**

Tic-tac-toe is a paper-and-pencil game for two players who take turns marking the spaces in a three-by-three grid with X or O. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner.

**Write a class/Type/Data Type to play this game**

**Code**

**class** **TicTacToe**:

**def** \_\_init\_\_(self):

self.board = [-1] \* 9 *# Using a 1D list to represent the 3x3 board*

self.current\_player = 0 *# Player 0 (X) goes first*

**def** play\_ttt(self):

win = **False**

move = 0

**while** **not** win **and** move < 9: *# The game ends after 9 moves*

self.print\_board()

**if** self.current\_player == 0:

turn = 'X'

**else**:

turn = 'O'

print(f"Turn for player **{**turn**}**")

user = self.get\_input()

**while** self.board[user] != -1:

print("Invalid Input. Try again.")

user = self.get\_input()

self.board[user] = self.current\_player

move += 1

**if** move > 4: *# A player needs at least 5 moves to win*

winner = self.check\_win()

**if** winner != -1:

self.print\_board()

print(f"The winner is **{**'X' **if** winner == 0 **else** 'O'**}**!")

**return**

self.current\_player = 1 - self.current\_player *# Switch player*

self.print\_board()

print("It's a draw!")

**def** get\_input(self):

**while** **True**:

**try**:

user = int(input("Enter a position (0-8): "))

**if** 0 <= user <= 8:

**return** user

**else**:

print("Invalid input. Enter a number between 0 and 8.")

**except** **ValueError**:

print("Invalid input. Enter a number between 0 and 8.")

**def** check\_win(self):

win\_cords = ((0, 1, 2), (3, 4, 5), (6, 7, 8), *# Rows*

(0, 3, 6), (1, 4, 7), (2, 5, 8), *# Columns*

(0, 4, 8), (2, 4, 6)) *# Diagonals*

**for** cord **in** win\_cords:

a, b, c = cord

**if** self.board[a] == self.board[b] == self.board[c] != -1:

**return** self.board[a]

**return** -1

**def** print\_board(self):

**for** i **in** range(0, 9, 3):

print(f"**{**self.display\_symbol(self.board[i])**}** | **{**self.display\_symbol(self.board[i + 1])**}** | **{**self.display\_symbol(self.board[i + 2])**}**")

**if** i < 6:

print("-" \* 9)

@staticmethod

**def** display\_symbol(player):

**if** player == 0:

**return** 'X'

**elif** player == 1:

**return** 'O'

**else**:

**return** ' '

**Results/Output**

|  |  |  |
| --- | --- | --- |
| **| |**  **---------**  **| |**  **---------**  **| |**  **Turn for player X**  **Enter a position (0-8):** | **Enter a position (0-8): 0**  **X | |**  **---------**  **| |**  **---------**  **| |** | **Turn for player O**  **Enter a position (0-8): 1**  **X | O |**  **---------**  **| |**  **---------**  **| |**  **Turn for player X** |
| **Initial state** | **X Player Enter 0 position** | **O Player Enter 1 position** |
| **Turn for player X**  **Enter a position (0-8): 3**  **X | O |**  **---------**  **X | |**  **---------**  **| |** | **Turn for player O**  **Enter a position (0-8): 4**  **X | O |**  **---------**  **X | O |**  **---------**  **| |** | **Turn for player X**  **Enter a position (0-8): 6**  **X | O |**  **---------**  **X | O |**  **---------**  **X | |**  **The winner is X!** |
| **X Player Enter 3 Position** | **O Player Enter 4 Position** | **X Player enter 6 position and win** |

1. **Conclusion**

이번 과제를 통해 파이썬의 다양한 데이터 구조와 기본 자료구조 개념을 심도 있게 이해하고 활용하는 경험을 얻었습니다.

**Python 데이터 구조 활용:**

이 과제에서는 Python 클래스를 사용하여 여러 데이터 구조를 구현하고 활용하는 방법을 배웠습니다. 특히, Matrix 클래스에서는 행렬 연산을 구현하고 EightQueens 클래스에서는 백트래킹을 활용하여 문제를 해결했습니다.

각 클래스별로 해당 데이터 구조를 선택한 이유는 다음과 같습니다:

- Matrix 클래스: 행렬 연산을 구현하기 위해 2차원 리스트를 사용하였고, 이는 행렬의 각 원소를 효과적으로 표현하기에 적합한 데이터 구조입니다.

- EightQueens 클래스: 여왕 8개를 배치하는 문제를 해결하기 위해 1차원 리스트를 사용하였고, 백트래킹 알고리즘을 활용하여 문제를 효과적으로 해결하였습니다.

- TicTacToe 클래스: 3x3 보드를 표현하기 위해 1차원 리스트를 사용하였고, 이는 간단한 틱택토 게임을 구현하기에 적합한 데이터 구조입니다. 이는 2차원 리스트를 사용했을 때 보다 훨씬 적은 메모리를 사용하여 구현할 수 있습니다.

저희는 리스트, 랜덤 모듈 등 Python의 내장 데이터 구조와 모듈을 적극적으로 활용했습니다. 특히, 리스트를 사용하여 행렬 및 체스 보드를 효과적으로 표현하고 조작하는 방법을 익혔습니다.

각 클래스 내에서 사용한 메서드들은 해당 클래스의 목적과 요구 사항을 따랐습니다. 예를 들어, Matrix 클래스의 `\_\_add\_\_`, `\_\_sub\_\_`, `\_\_mul\_\_` 메서드는 행렬 연산에 필요한 메서드로 구현되었습니다.

위처럼 python special method 메서드 오버라이딩을 사용하면 객체를 더 읽기 쉽고 이해하기 쉽게 표현할 수 있습니다. 예를 들어, Matrix 클래스의 `\_\_str\_\_` 메서드를 오버라이딩하여 `print` 함수로 행렬을 직접 출력할 때 객체의 내용이 아닌 메모리 위치가 표시되지 않도록 수정했습니다.

**느낀점**

이번 과제를 통해 Python의 데이터 구조와 객체 지향 프로그래밍에 대한 이해가 상당히 높아졌습니다. 각 클래스를 구현하고 문제를 해결하기 위해 어떤 데이터 구조를 선택해야 하는지에 대한 고려가 필수적이었습니다. 이러한 선택은 컴퓨팅 파워와 성능에 상당한 영향을 미칠 수 있으며, 실제 엔지니어로서 이런 기술을 함양하는 것이 중요하다는 것을 명확하게 깨달았습니다.

특히, Python의 내장 데이터 구조와 모듈을 활용하여 문제를 해결하는 방법을 배웠습니다. 이러한 내장 도구들은 코드를 간결하게 작성하고 문제에 대한 효율적인 솔루션을 개발하는 데 매우 유용했습니다.

또한, 요구사항을 바탕으로 소스코드를 작성하는 것을 경험함으로, 요구사항의 의도를 명확히 파악하고 개발하는 것에 대해서 점점 익숙해 지기 시작했습니다.

이러한 경험은 프로그래밍 능력을 향상시키는 데 큰 도움이 되었고, 적합한 자료구조 선택 능력을 기를 수 있는 기회였습니다.