Problem 3 : N-queen

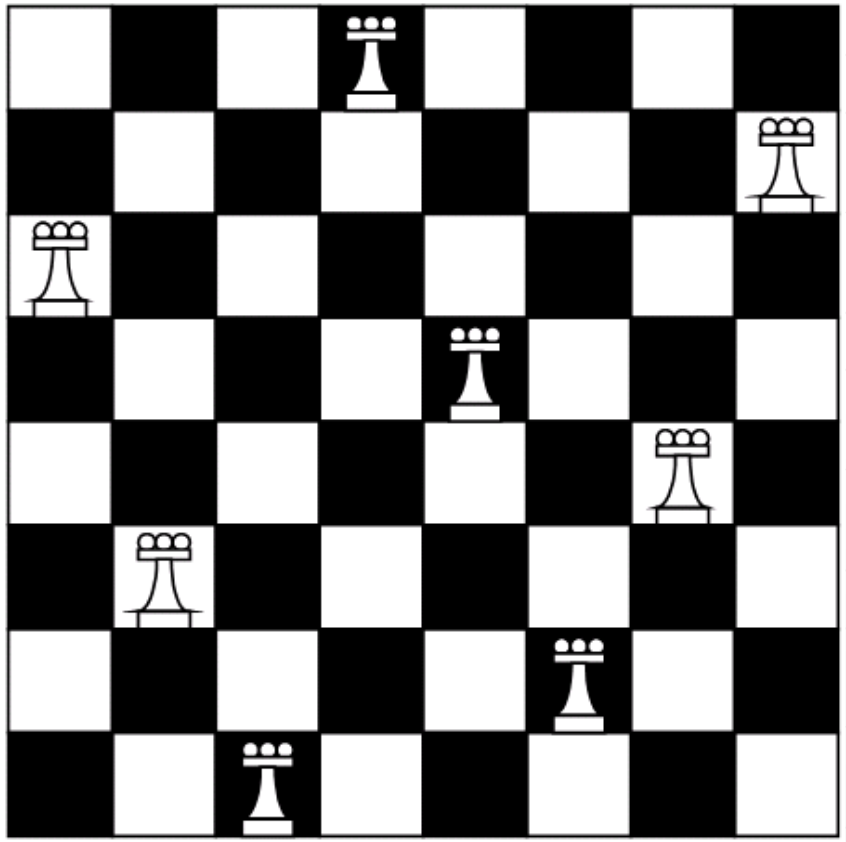
17101992 Hong Sumin

Agenda

- N-queen
- Stack & Backtracking
- Design
- Stack ver.
- Additional : Recursion ver.
- Q&A

N-queen

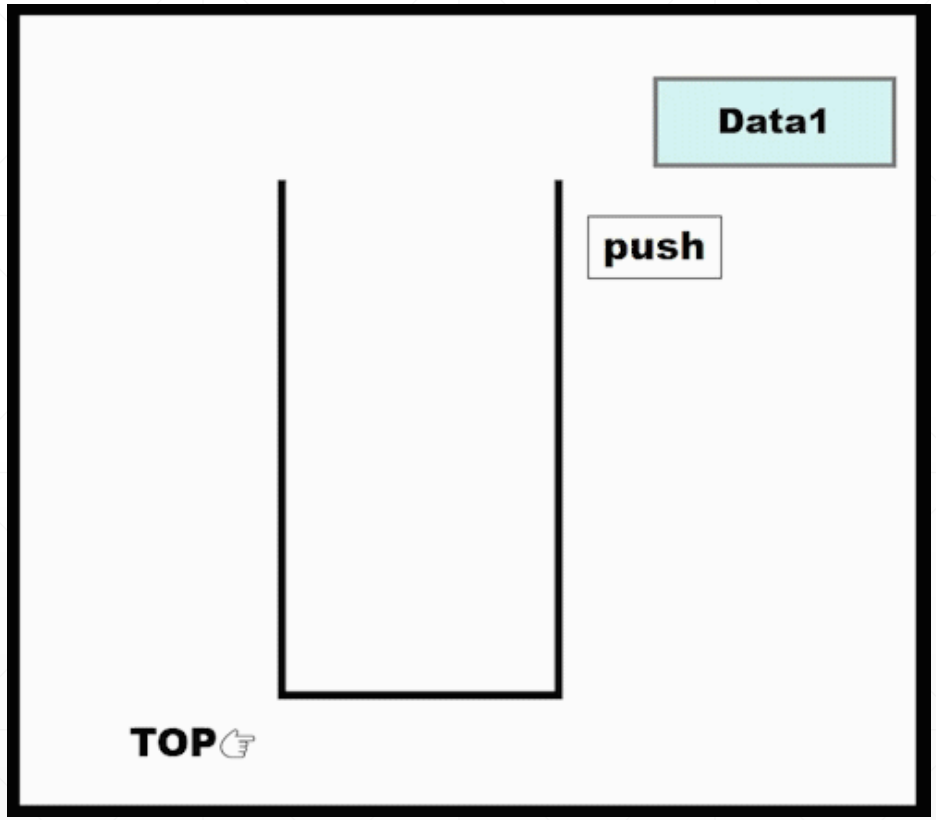
- N-queen



- In N x N size Chess Board, place N queens against so that they can’t attack each other.
- Queen can go straight up and down, diagonally without limits.

Stack & BackTracking

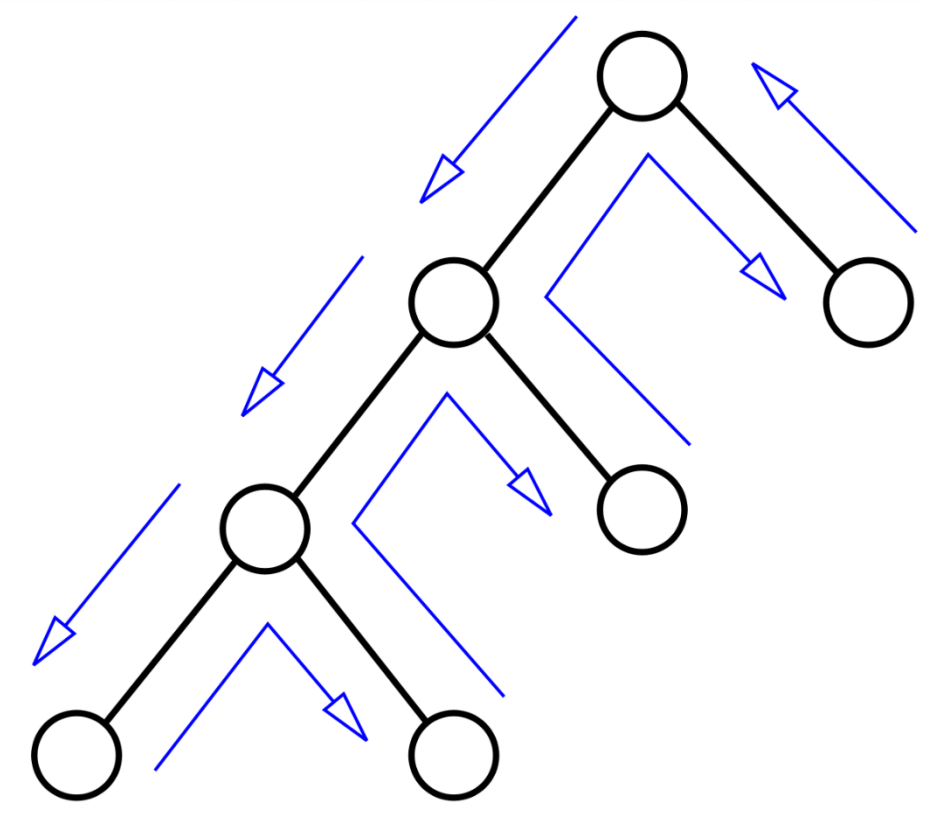
- Stack



- Stack is a data structure which has LIFO(Last –in, First-out) format.
- We can push and pop data at one-side.

Stack & BackTracking

- BackTracking



- Systematically searching the solution, notably constraint satisfaction problems.
- Incrementally builds candidates to the solutions

Stack ver.

```python
class Stack(object):
    def __init__(self):
        self.items = []
        self.maxsize = 0
        self.size = 0

    def push(self, value):
        self.items.append(value)

    def pop(self):
        try:
            val = self.items.pop()
            return val
        except(IndexError):
            print("Stack is empty.")

    def peek(self):
        if self.items:
            return self.items[-1]
        else:
            print("Stack is empty.")

    def isEmpty(self):
        return not bool(self.items)
```

```python
def isFull(self):
    if self.getStackSize() == self.maxsize:
        return True
    else:
        return False

def getStackSize(self):
    self.size = len(self.items)
    return self.size

def printStack(self):
    if self.items:
        print(self.items)
    else:
        print("Stack is empty.")

def getSizeOfChs(self):

    while True:
        try:
            n = int(input("Please input the chessboard size >>"))
            if not 4 <= n <= 8:
                raise ValueError
            else:
                self.maxsize = n
                break
        except(ValueError):
            print("Input right value.")
            pass
```

- Stack class
  - Attributes
    - Items
    - Maxsize
    - Size
  - Method
    - isEmpty : determine stack is empty or not
    - isFull : determine stack is full or not
    - getStackSize : get stack(chess board) size
    - printStack : print stack components
  - getSizeOfChs : Get size of chessboard to user. If it's size isn't 4~8, rasie Value Error. Return size of chessboard if it is right value.

Stack ver.

```python
def findPosition(stack):

    if stack.isEmpty():
        pstn = [1, 1]
        stack.push(pstn)
        notEmpty = True
    else:
        notEmpty = True

    # for i in range(stack.size, stack.max+1):

    while notEmpty:

        if stack.isFull():
            return True # return값 다시 설정 필요
            break

        raw = stack.getStackSize()+1
        # stack.printStack()

        for i in range(stack.maxsize): #한 행에서 4번 검사 -> 어떤 행인지 알려줄 필요 있음
            # print("현재 인덱스",raw, i+1)
            idx = [raw, i+1]

            bool = verify(stack, idx) #수정
            # print("검증 결과",bool)
            if bool:
                stack.push(idx)
                break  # for문 탈출

        if not bool:
            # print("There is no valid position. Start Backtracking.")
            stack = backTracking(stack)
```

- findPosition
  - When start, push (1, 1) to stack
  - When stack is full(find all queen's place), return True and break while statement.
  - In one raw, we have to check N times.
  - Put stack & index to verify method.
  - If index is valid, push it to stack and break for statement.
  - Else(nowhere valid in on raw), call backtracking function.

Stack ver.

```python
def backTracking(stack):

    while True:
        pop = stack.pop()
        stack.printStack()

        if stack.isEmpty():
            # print("실행되니?")
            idx = [pop[0], pop[1]+1]
            stack.push(idx)
            return stack
            break

        elif pop[1]!=stack.maxsize:
            idx = [pop[0], pop[1]]
            for i in range(pop[1]+1,stack.maxsize+1):#pop한 좌표 다음 좌표부터 집어넣기
                idx[1]+=1
                # print("인덱스 더함", idx)
                bool = verify(stack, idx)
                # print("검증결과", bool)
                if bool: #for문 탈출
                    stack.push(idx)
                    break

            if bool: #while문 탈출
                return stack
                break

        else:
            pass # 스택 하나 더 pop
```

- backtracking
  - Pop last component
  - If Stack is empty(pop first raw queen), push index that next column of pop data.
  - Else if pop data is in last column, pop next data of stack (have to change low level data)
  - Else, verify other columns, start at next column of the pop data

Stack ver.

```python
def verify(stack, idx):
    # 인덱스 받아서 열 및 대각선에 여왕이 있는지 검사
    # 없으면 True 아니면 False
    flg = None
    for raw, col in stack.items:
        # print("스택 인덱스 열",raw, col)
        if idx[1] == col:
            flg = False
            break

        elif (col-idx[1])/(raw-idx[0]) == -1 or (col-idx[1])/(raw-idx[0]) == 1:
            # print("기울기",(col-idx[1])/(raw-idx[0]))
            flg = False
            break

        else:
            flg = True

    return flg
```

- Verify
  - Verify queen's position is okay
  - Get stack items' raw & column, check is any queen in the column.
  - And Check slope of stack's queen and index. If 1 or -1, It isn't valid position.
  - Return True if it is right index.

Stack ver.

```
def printChessBoard(stack):

    mat = [["|     |"] * (stack.maxsize+1) for i in range(stack.maxsize)]

    for raw, col in stack.items:

        mat[raw-1][0] = "Row %d - Col %d" %(raw, col)
        print(raw-1, col)
        mat[raw-1][col] = ("|  Q  |")

    print(tabulate(mat))
```

- printChessBoard
  - Create a two-dimensional list with size of a chessboard +1 column that filled with space.
  - Change first column to scripts that explain queen's place
  - Mark the queen's place at table
  - Use tabulate module to print 2-D list.

Stack ver.

```
if __name__ == '__main__':
    timelist = []
    for i in range(10):
        start = time.time()

        ChsBrd = Stack()
        ChsBrd.maxsize = getSizeOfChs()
        findPosition(ChsBrd)




        printChessBoard(ChsBrd)

        timelist.append(time.time() - start)

    print(sum(timelist), len(timelist))
    print(sum(timelist) / len(timelist))
```

```
Please input the chessboard size >>4
-------------  -------  -------  -------  -------
Row 1 - Col 2  |     |  |  Q  |  |     |  |     |
Row 2 - Col 4  |     |  |     |  |     |  |  Q  |
Row 3 - Col 1  |  Q  |  |     |  |     |  |     |
Row 4 - Col 3  |     |  |     |  |  Q  |  |     |
-------------  -------  -------  -------  -------
```

- Main
  - Use Stack class to make chessboard stack
  - Get chessboard's size using getSizeOfChs
  - FindPosition & Print Chessboard
- Print Result

Recursion ver.

```
findPosition(ChsBrd, [1,1])
```

```
def findPosition(stack, idx):

    if stack.isEmpty():
        stack.push(idx)
        nidx = [idx[0] + 1, idx[1]]
        # print(nidx)
        return findPosition(stack, nidx)

    if stack.isFull():
        return True # return값 다시 설정 필요
```

- findPosition(Recursion)
  - 1. Stack is Empty(At start)
    - Push [1, 1] & go to next index
    - Return findPosition with new index
  - 2. Stack is Full(At end)
    - Return True (Recursion is end!)

Recursion ver.

```
# stack.printStack()
if idx[1] > stack.maxsize:
    # print("어떤 열에도 여왕을 놓을 수 없음, pop해야함")
    pop = stack.pop()
    # print("pop", pop)
    nidx = [pop[0], pop[1]+1]
    return findPosition(stack, nidx)

elif verify(stack,idx):
    # print("여왕을 넣음", idx)
    stack.push(idx)
    nidx = [idx[0]+1, 1]
    return findPosition(stack, nidx)

elif not verify(stack,idx):
    nidx = [idx[0], idx[1]+1]
    # print("다음 열로 넘어감!")
    return findPosition(stack, nidx)
```

- findPosition(Recursion)
  - 3. column of the index is at the last column
    - In that raw, there is no valid place
    - Pop stack & set new index to next column of pop
    - Return findPosition
  - 4. The index is valid place
    - Push index
    - Set new index to next raw
    - Return findPosition
  - 5. The index is not valid place
    - Set new index to next column of pop
    - Return findPosition

Comparison two version By Time

```
1.4713928699493408 100000
1.4713928699493408e-05

Process finished with exit code 0
```

Stack ver.

```
2.5860090255737305 100000
2.5860090255737305e-05

Process finished with exit code 0
```

Recursion ver.

- Get average of 100,000 times.
- Fixed chessboard size at 4x4.
- Recursion version takes longer(1.7times) than Stack version

Time Comparison By Size : Stack ver.

1.4713928699493408 100000 1.4713928699493408e-05	4x4.
0.8188326358795166 100000 8.188326358795166e-06	5x5
10.347476720809937 100000 0.00010347476720809937	6x6
2.608022928237915 100000 2.608022928237915e-05	7x7
60.61560320854187 100000 0.0006061560320854187	8x8
23.765286445617676 100000 0.00023765286445617676	9x9

- Get average of 100,000 times.
- Size of chess board & computation time doesn't grow linearly.
- But 8x8 size takes overwhelmingly long hours!
- Odd number is about half shorter. Why????

Time Comparison By Size : Recursion ver.

2.5631582736968994 100000 2.5631582736968993e-05	4x4.
1.5000357627868652 100000 1.5000357627868653e-05	5x5
21.203966856002808 100000 0.00021203966856002808	6x6
5.152649641036987 100000 5.1526496410369876e-05	7x7
135.99912285804749 100000 0.0013599912285804475	8x8
51.46688795089722 100000 0.0005146688795089722	9x9

- Get average of 100,000 times.
- Size of chess board & computation time doesn't grow linearly
- But 8x8 size takes overwhelmingly long hours, too!
- Odd number is shorter, too.

Comparison two version By Memory

47.70000000008992

Stack ver.

47.79999999990988

Recursion ver.

48.60000000000571

49.0

- Use psutil module.
- Get average of 100,000 times.
- Fixed chessboard size at 4x4.
- Both versions have similar memory usage.

Size Comparison By Memory

47.70000000008992	4x4.
48.60000000000571	5x5
48.5	6x6
48.60000000000571	7x7
49.0	8x8

- Use psutil module.
- Executed on stack version.
- Get average of 100,000 times.
- No significant difference than time.

Q&A

https://colab.research.google.com/drive/1d6FVBJqtAzQMsWAULUa8_eUAmB_uAy21?usp=sharing