5118006-03 Data Structures

# Stack

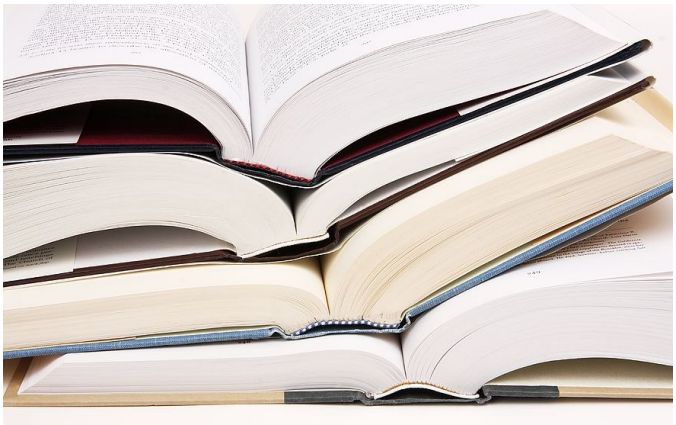
20 Mar 2024

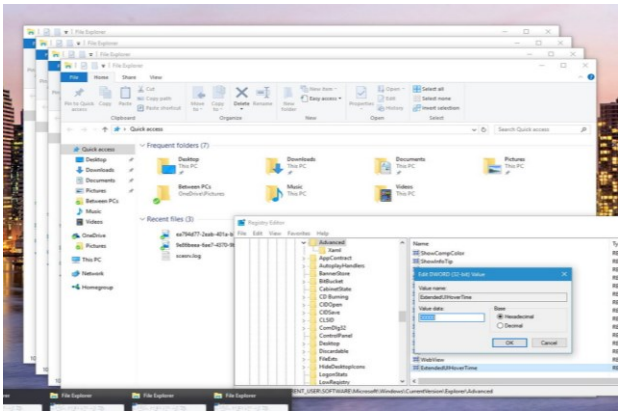
Shin Hong

# Stack

- A stack is an ordered list where insertion and deletion is made only at one end
  - stack is also called LIFO (Last-In-First-Out)
  - the end to which an item is inserted/deleted is called top

- A stack is useful for storing temporal states in recursive search



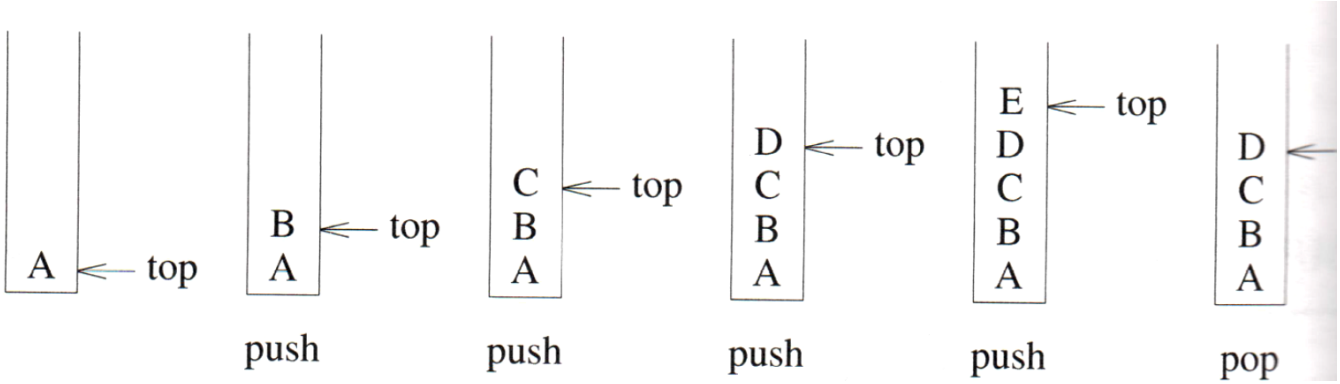


# Abstract Data Type

- Data container
  - **buffer**: an array to hold elements
  - **capacity**: the capacity of the buffer array
  - **top**: an index of the array to place a next element if the buffer is not full, or the capacity of the buffer

- Operations
  - **push(e)**:  insert a new element **e** to the stack if the stack is not full
  - **pop()**: return and remove the most recently inserted element if the stack is not empty
  - **top()**: return and remove the most recently inserted element if the stack is not empty
  - **isEmpty()** : return whether the stack has at least one element or not
  - **isFull()** : return whether the stack is full or not

# Example


A
top
B
A
top
push
C
B
A
top
push
D
C
B
A
top
push
E
D
C
B
A
top
push
D
C
B
A
pop

# Implementation

- https://github.com/hongshin/DataStructures/tree/main/code/Mar22

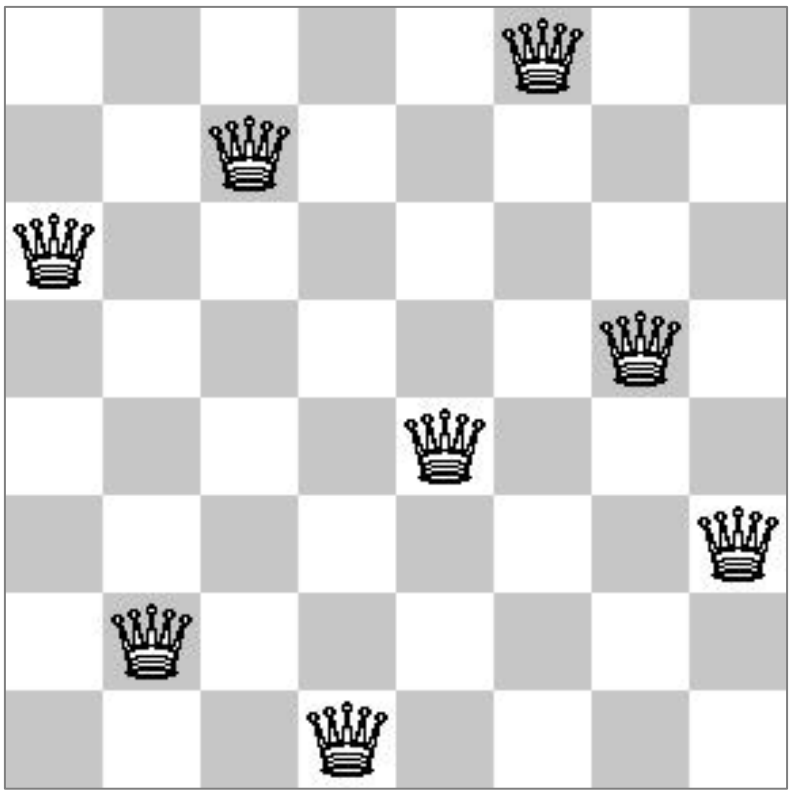
# Use of Stack

- A stack is used for storing a series of decisions made in a solution search

- Ex. checking if nested brackets are balanced
  - A given string of brackets is balanced if a left side bracket (opening) is immediately followed by the corresponding right side (closing), or followed by another balanced bracket and then the right side.
  - Ex.
    - `(((()())()))`
    - `((()))(())`
    - `([{]})`
    - `([]{()[]}){[]}`

# Case 1. N-Queens Problem

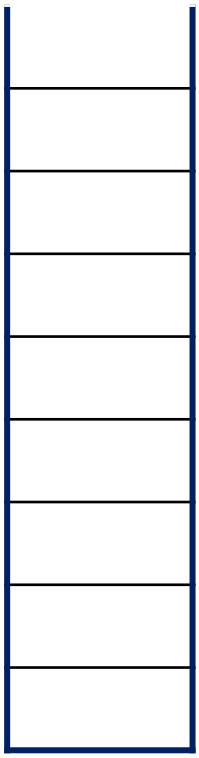
- Find a placement of N queens on a checkboard such that they do not conflict with each other
  - Two queens cannot stand together if they are on the same vertical / horizontal / diagonal line



# Case 1. N-Queens Problem

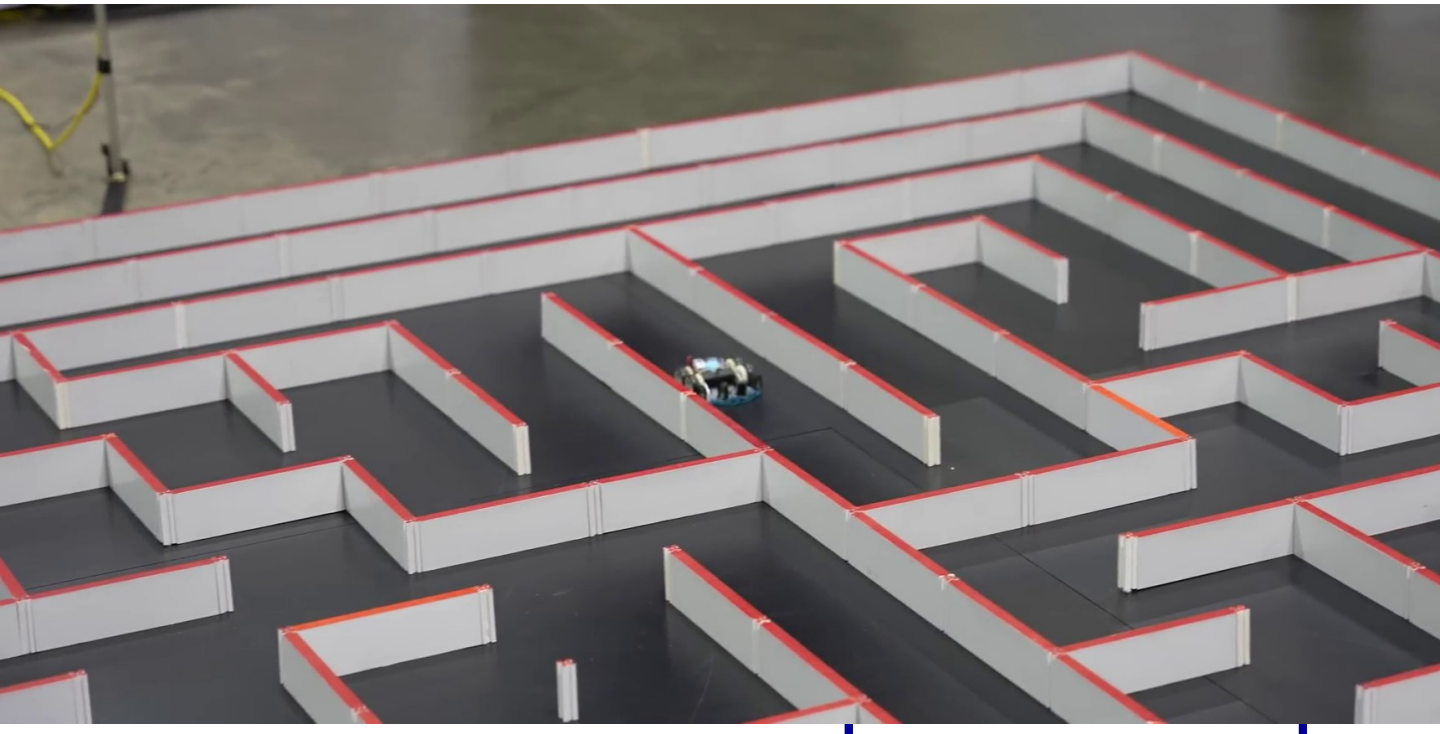
- Find a placement of N queens on a checkboard such that they do not conflict with each other
  - Two queens cannot stand together if they are on the same vertical / horizontal / diagonal line

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
| 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |



# Case 2. Maze

- Find a path that consists of vertical and/or horizontal lines from the top-left corner (entrance) to the bottom-right corner
  - a player can move **up**, **down**, **left** or **right** to an empty cell
- Store the current path in a stack
  - each element represents the exploration status at a cell



# Case 3. Evaluating Expression

- An expression is a value, or one or more expressions connected with an operator

- Different notation to represent an arithmetic expression
  - Postfix: an operator is placed after its operands
  - Prefix: an operator is placed before its operands
  - Infix: a binary operator is placed between two operands
    - ambiguity

- Example
  - Postfix: `3 6 + 2 4 - * 7 +`
  - Prefix: `+ * + 3 6 - 2 4 7`
  - Infix : `((3 + 6) * (2 - 4)) + 7`