Data Structure

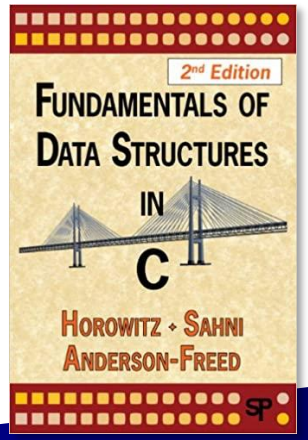
Stack

for Backtracking and Exploring Hierarchical Data

Shin Hong

Mar 31, 2020

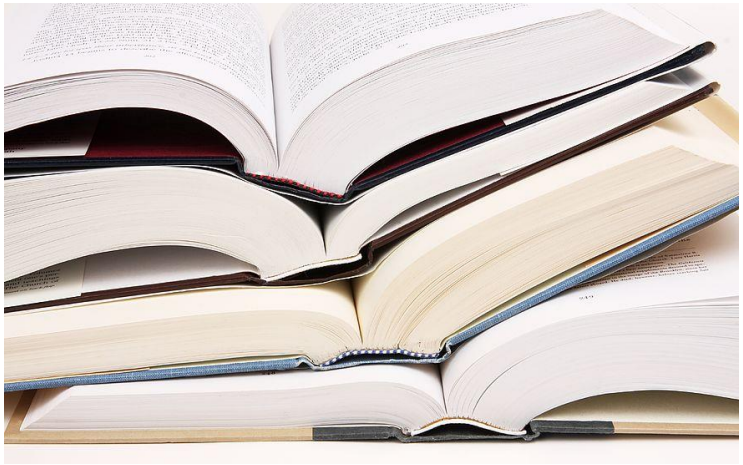
Chapter 3.
Stack and Queue

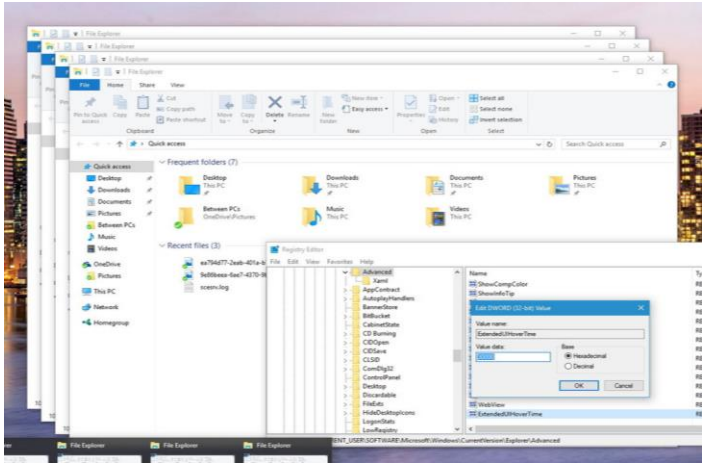


Stack

- A stack is a list where insertion and deletion is made only at one side of the end (top)
  - a stack is also called as LIFO (Last-In-First-Out)

- A stack is useful for storing a temporal state of a search on a hierarchical structure



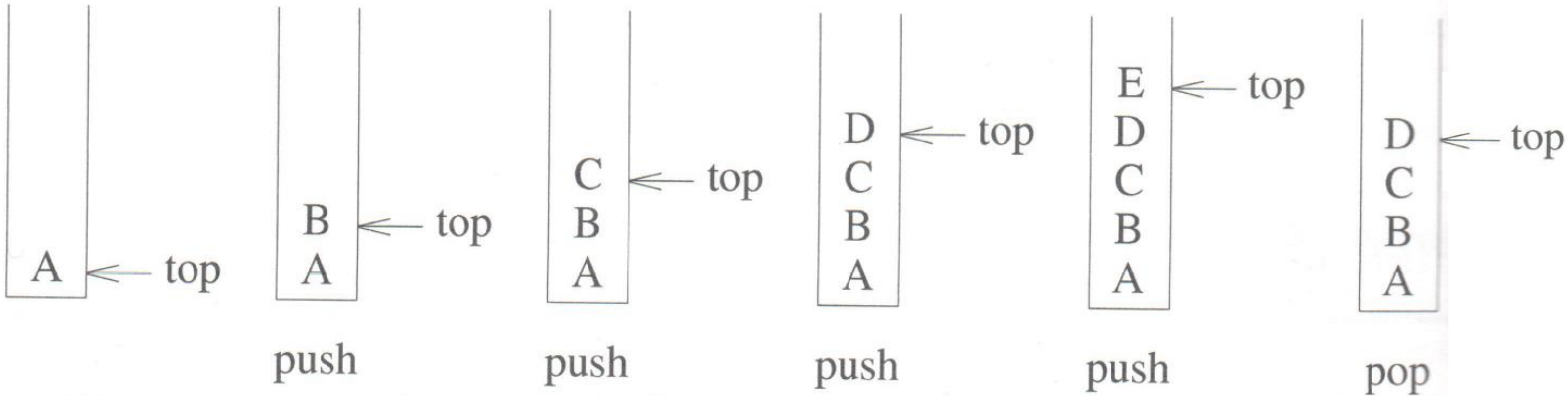


Stack Abstract Data Type

- Structure
  - **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 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
top
pop

Implementation

- Stack for integers
  - see https://github.com/hongshin/DataStructure/tree/stack/ver1

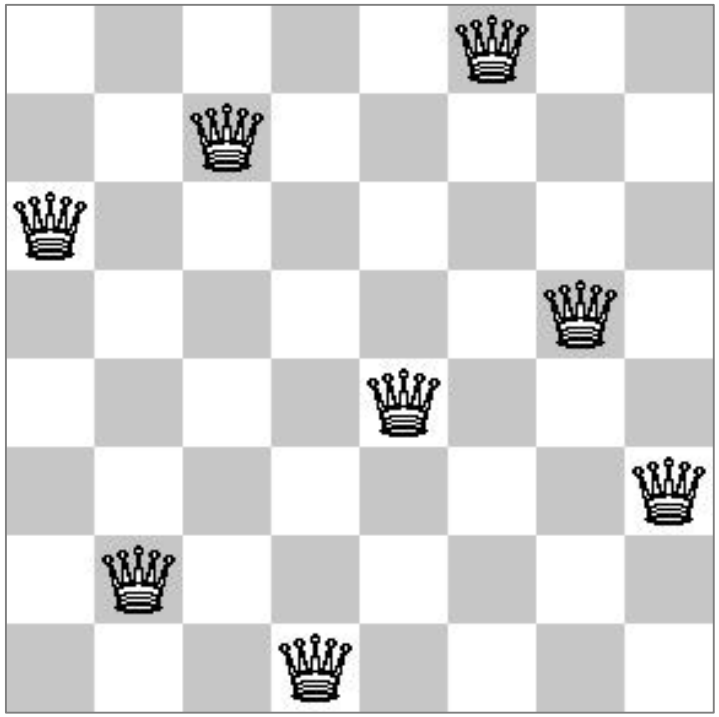
- How to construct a stack for all element types?
  - see https://github.com/hongshin/DataStructure/tree/stack/ver2

Backtracking

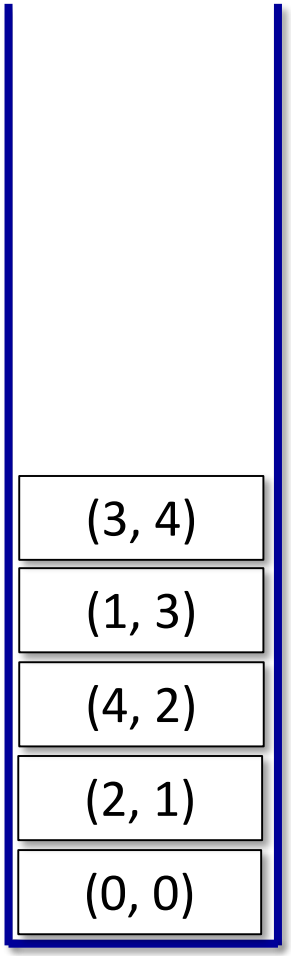
- There is a problem whose solution is a combination of (small) decisions
- A backtracking is a strategy to enumerate all possible solutions by recursively exploring all decision sequences
  - E.g., breaking a dial lock
- A stack is useful to represent the current status of the solution (a sequence of decisions) in backtracking

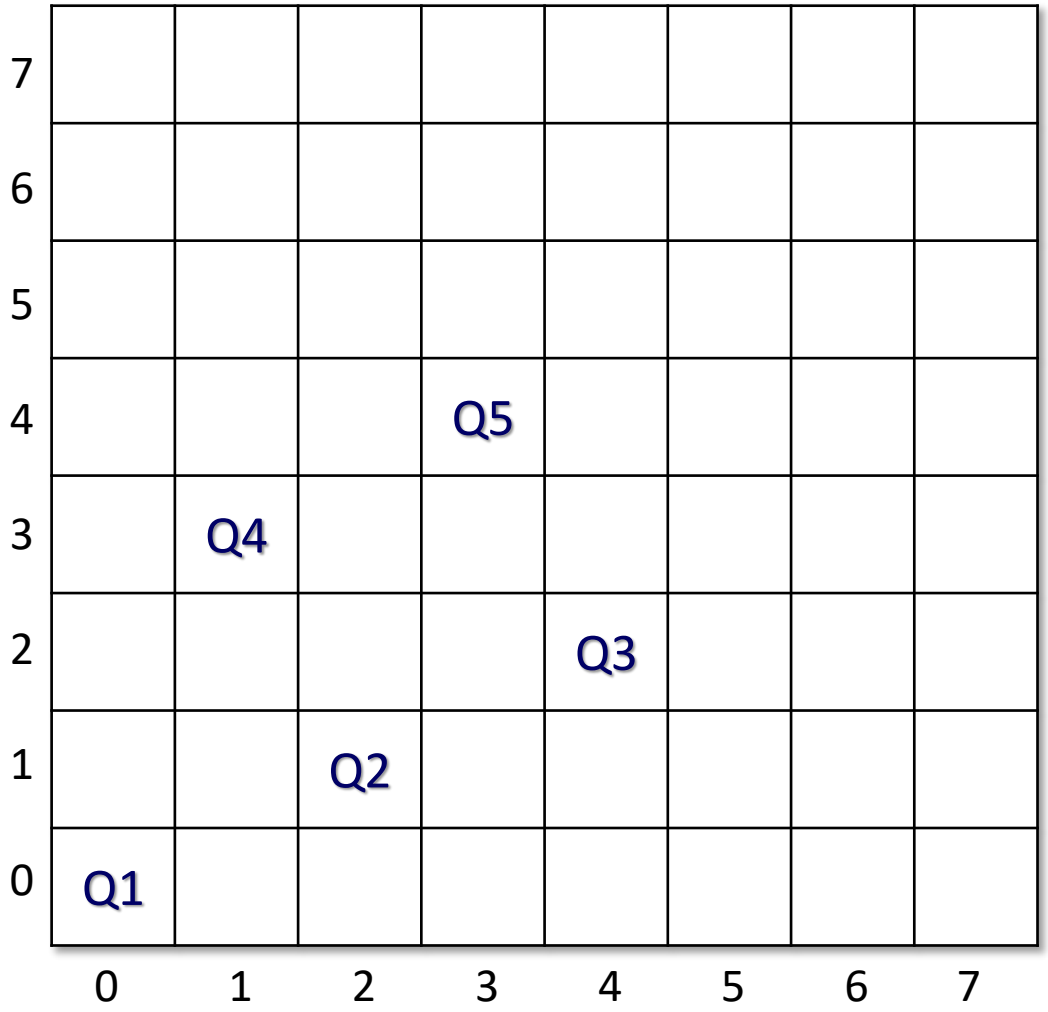
Case 1. N Queen Problem

- 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



Backtracking Queen Placement

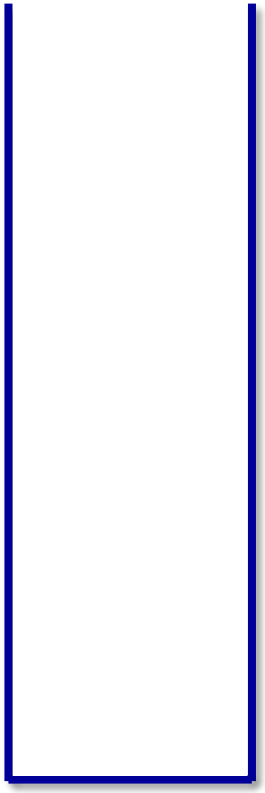


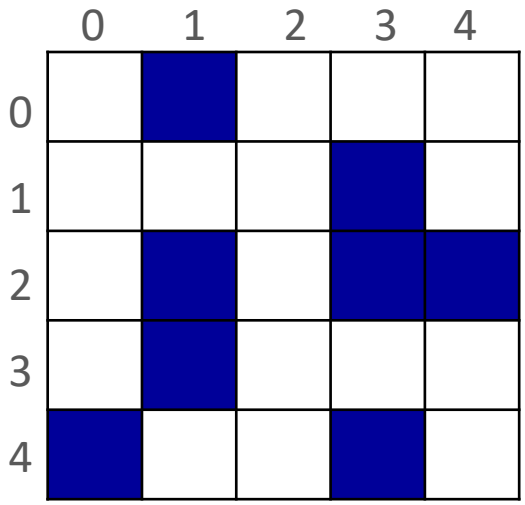


No place for Q6

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 stack elemenet reprensets the exploration status at a cell





Case 3. Evaluating Expression

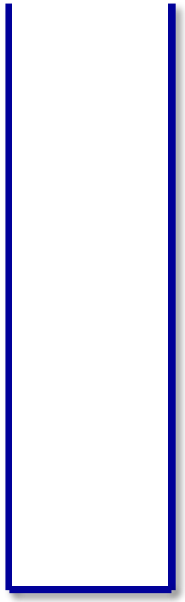
- An expression is a value, or one or more expressions connected with an operator

- Different notation to represent an arithemetic 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`

Evaluating Postfix Expression

- Store operands in the stack for next operator
  - top-1 and top-2 are the operands for next operator
- Push the evaluation result back to the stack for next operator
- Example: `3 6 + 2 4 - * 7 +`



Evaluating Prefix Expression

- Store an operator and first operand value in the stack until second operand value comes
- Example: `+ * + 3 6 - 2 4 7`