DATA STRUCTURE AND ALGORITHMS

LECTURE 4

Stack, Queue

DATA STRUCTURE AND ALGORITHMS

LECTURE 4a

Stack ADT

Reference links:

https://cs.nyu.edu/courses/fall17/CSCI-UA.0102-007/notes.php

https://www.comp.nus.edu.sg/~stevenha/cs2040.html

[M.Goodrich, chapter 6, sec 6.1]

Lecture outline

- Stack ADT
  - Introduction
  - Specification
  - Implementations
    - Array Based
    - Linked List Based
  - Applications
    - Bracket Matching
    - Tower of Hanoi
    - Maze Exploration

Stack ADT

Introduction

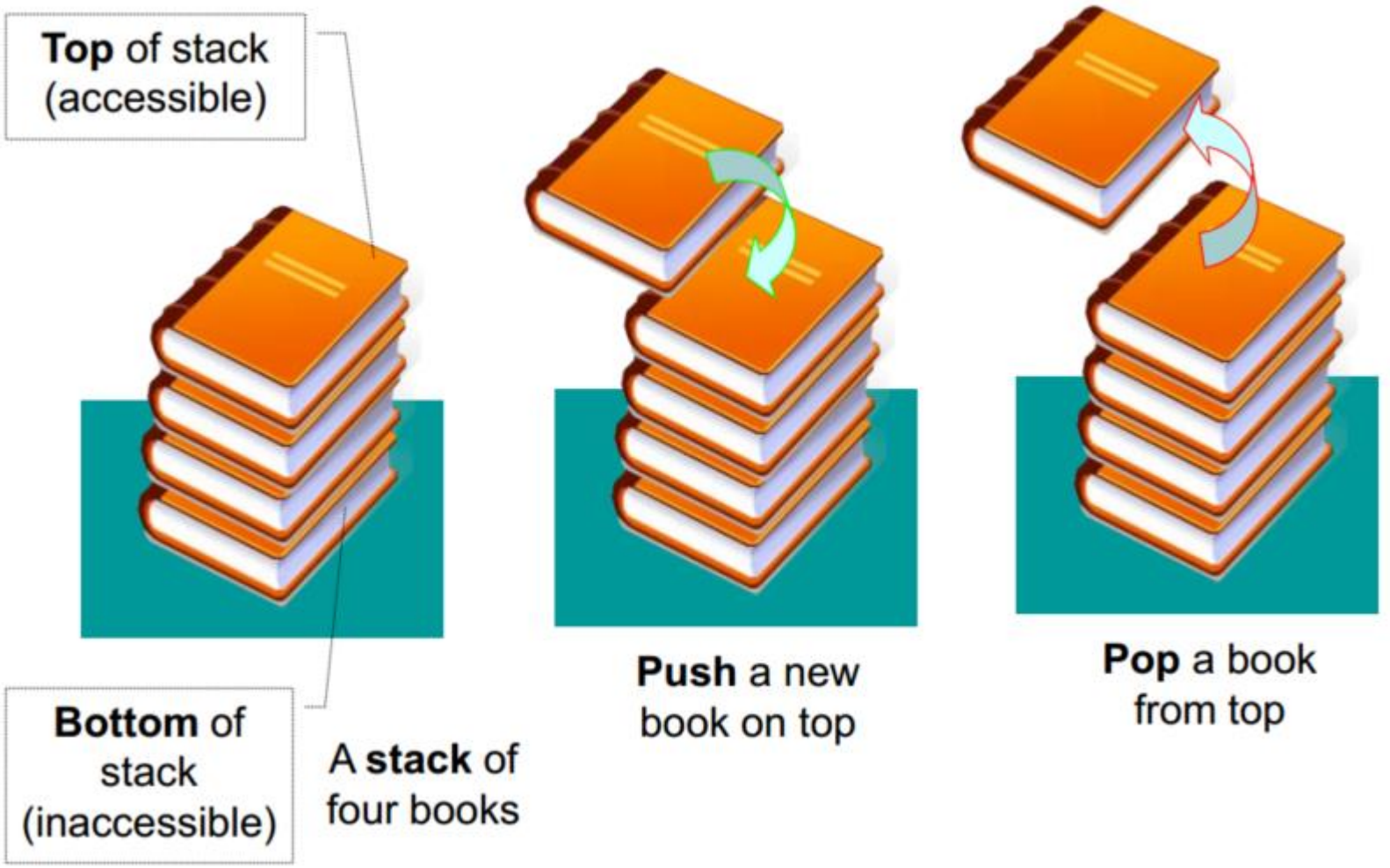
Stack: A specialized list

- List ADT (lecture 3b) allows user to manipulate (insert/retrieve/remove) item at **any position within the sequence of items**
- There are cases where we only want to consider a few specific positions only
  - E.g. only the first/last position
  - Can be considered as special cases of list
- **Stack** (this lecture – 4a) is one such example
  - Only manipulation at the last position is allowed
- **Queue** (lecture 4b) is another example
  - Only manipulation at the first and last position are allowed

What is a stack

- Real life example:
  - A stack of books, A stack of plates, Etc..
- It is easier to add/remove item to/from the **top of the stack**
- The latest item added is the first item you can get out from the stack
  - Known as **L**ast **I**n **F**irst **O**ut (**LIFO**) order
- Major Operations:
  - Push : Place item on top of the stack
  - Pop : Remove item from the top of the stack
  - Top : Take a look at the topmost item without removing it

Stack: Illustration



Source: Visualgo.net

Stack ADT

Specification

Stack specification in Java

```
public interface Stack<E> {
        //Returns the number of elements in the stack
        int size();
        //Tests whether the stack is empty
        public boolean isEmpty();
        //Inserts an element at the top of the stack
        public void push(E element);
        //Returns, but does not remove, the element at the top
        public E top();
        //Removes and returns the top element from the stack
        public E pop();
}
```

A simple version of the stack interface [M.Goodrich,229]

Stack specification in Java

- Class Stack in java.util
  - https://docs.oracle.com/javase/9/docs/api/java/util/Stack.html

Stack ADT

Implementation

Stack: Implementations

- Some ways to implement Stack ADT, we will cover:
  - Array implementation
  - Linked List implementation

- Learn how to weight the pros and cons for each implementations – điểm mạnh điểm yếu của mỗi cấu trúc cài đặt

Stack: Implementations

Using Array

[M. Goodrich, sub-section 6.1.2]

Stack Implementation using array

```
1  public class ArrayStack<E> implements Stack<E> {
2    public static final int CAPACITY=1000;  // default array capacity
3    private E[ ] data;                      // generic array used for storage
4    private int t = −1;                     // index of the top element in stack
5    public ArrayStack( ) { this(CAPACITY); } // constructs stack with default capacity
6    public ArrayStack(int capacity) {       // constructs stack with given capacity
7      data = (E[ ]) new Object[capacity];   // safe cast; compiler may give warning
8    }
9    public int size( ) { return (t + 1); }
10   public boolean isEmpty( ) { return (t == −1); }
11   public void push(E e) throws IllegalStateException {
12     if (size( ) == data.length) throw new IllegalStateException("Stack is full");
13     data[++t] = e;                        // increment t before storing new item
14   }
15   public E top( ) {
16     if (isEmpty( )) return null;
17     return data[t];
18   }
19   public E pop( ) {
20     if (isEmpty( )) return null;
21     E answer = data[t];
22     data[t] = null;                       // dereference to help garbage collection
23     t−−;
24     return answer;
25   }
26 }
```

Array-based implementation of the Stack interface [M.Goodrich, p230]

Stack Implementation using array

```
Stack<Integer> S = new ArrayStack<>();      // contents: ()
S.push(5);                                  // contents: (5)
S.push(3);                                  // contents: (5, 3)
System.out.println(S.size());               // contents: (5, 3)      outputs 2
System.out.println(S.pop());                // contents: (5)         outputs 3
System.out.println(S.isEmpty());            // contents: (5)         outputs false
System.out.println(S.pop());                // contents: ()          outputs 5
System.out.println(S.isEmpty());            // contents: ()          outputs true
System.out.println(S.pop());                // contents: ()          outputs null
S.push(7);                                  // contents: (7)
S.push(9);                                  // contents: (7, 9)
System.out.println(S.top());                // contents: (7, 9)      outputs 9
S.push(4);                                  // contents: (7, 9, 4)
System.out.println(S.size());               // contents: (7, 9, 4)   outputs 3
System.out.println(S.pop());                // contents: (7, 9)      outputs 4
S.push(6);                                  // contents: (7, 9, 6)
S.push(8);                                  // contents: (7, 9, 6, 8)
System.out.println(S.pop());                // contents: (7, 9, 6)   outputs 8
```

Sample usage of our ArrayStack class [M.Goodrich, p232]

Stack Implementation using array

- Analyzing the implementation
  - Running time

Method	Running Time
size	O(1)
isEmpty	O(1)
top	O(1)
push	O(1)
pop	O(1)

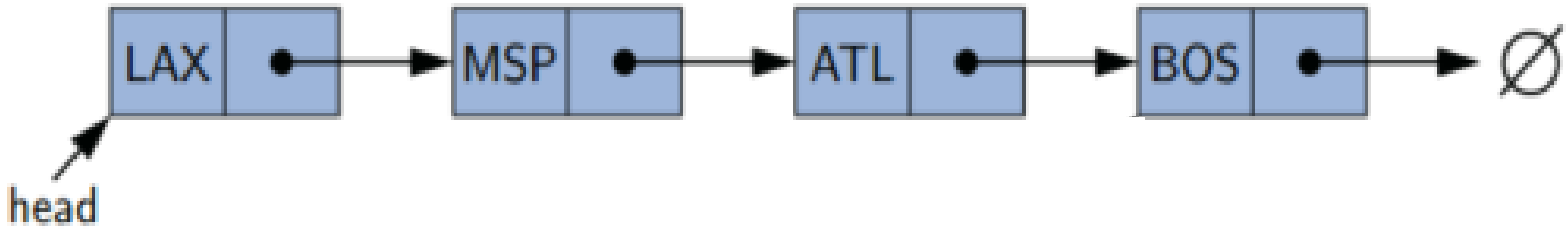
  - Storage: O(*N*) - *N* là kích thước mảng được khai báo

Stack: Implementations

Using Singly Linked List

[M. Goodrich, subsection 6.1.3]

Stack Implementation using linked list



- Characteristics of singly linked list
  - Efficient manipulation of 1st node:
    - Has a reference to it - tham chiếu linkedlist chính là phần tử đầu
    - No need to traverse the list – không cần duyệt danh sách
  - Without an arbitrary capacity limit – không giới hạn số phần tử
- Hence, use singly linked list for storing of stack
  - Use 1st node as the top of stack

Stack Implementation using linked list

- The corresponding methods from SinglyLinkedList

Stack Method	*Singly Linked List Method*
size( )	list.size( )
isEmpty( )	list.isEmpty( )
push(e)	list.addFirst(e)
pop( )	list.removeFirst( )
top( )	list.first( )

- Implements:

```
public class LinkedStack<E> implements Stack<E> {
  private SinglyLinkedList<E> list = new SinglyLinkedList<>();    // an empty list
  public LinkedStack( ) { }                  // new stack relies on the initially empty list
  public int size( ) { return list.size( ); }
  public boolean isEmpty( ) { return list.isEmpty( ); }
  public void push(E element) { list.addFirst(element); }
  public E top( ) { return list.first( ); }
  public E pop( ) { return list.removeFirst( ); }
}
```

Implementation of the Stack using SinglyLinkedList [M.Goodrich, p233]

Stack Implementation using linked list

- Analyzing the implementation
  - Running time – same as using array

Method	Running Time
size	O(1)
isEmpty	O(1)
top	O(1)
push	O(1)
pop	O(1)

  - Storage: O(*M*) - *M* là số nhiều nhất phần tử được lưu và xử lí trong stack

Stack ADT

Stack applications

Last In First Out!

LIFO – Is it good for anything?

Stack applications

- Bracket Matching

  Kiểm tra tính tương ứng cặp dấu ngoặc của biểu thức

- Tower of Hanoi

  Bài toán tháp Hà nội

- Exploring a Maze

  Khám phá mê cung

Stack application: Bracket Matching

- Mathematical expression can get quite convoluted:
  - E.g. { [x + 2(i - 4!)]^e + 4π/5 * (φ-7.28) ……
- We are interested in checking whether all brackets are matched correctly (with), [with] and { with }
- Bracket matching is equally useful for checking programming code

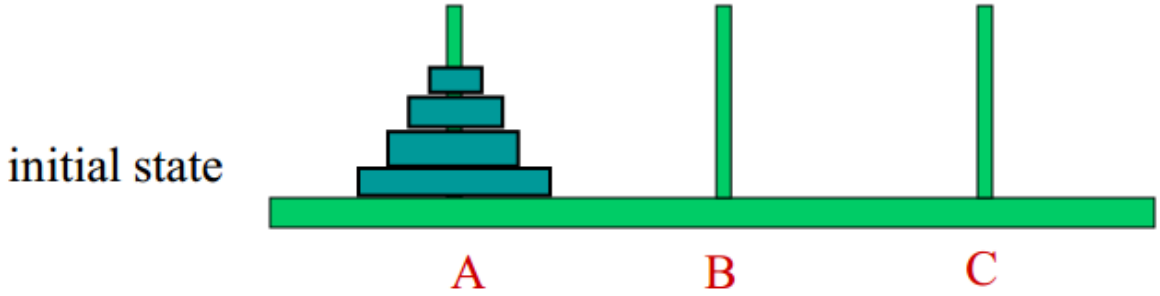
Bracket Matching: algorithm

- Idea and Algorithm: use a stack
  - Input: string - xâu biểu thức
  - Ouput: string is a valid expression – biểu thức hợp lệ?
  - Steps:

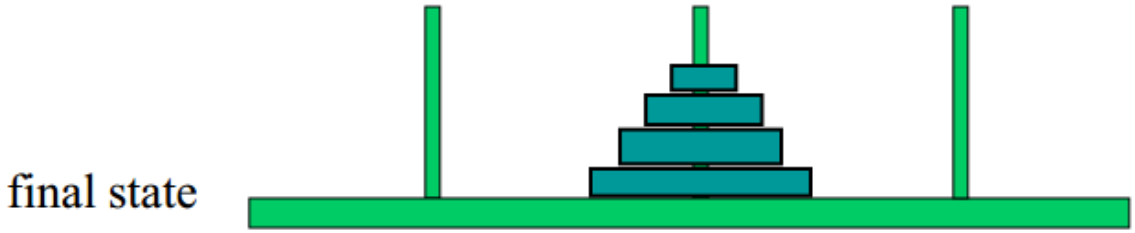
```
Duyệt từng kí tự trong xâu cần kiểm tra
    if (Không phải là kí tự dấu ngoặc)
        Bỏ qua;
    if (là kí tự ngoặc mở “{“ , “[“ or “(“)
        Cho vào stack (Push);
    if (là kí tự ngoặc đóng “}”, “]” or “)”)
        Lấy phần tử từ stack (Pop) và kiểm tra
        if (dấu ngoặc ở đỉnh không tương ứng với kí tự đang kiểm tra)
            “Báo biểu thức lỗi và kết thúc”;
        else continue;
if (duyệt hết xâu kí tự mà stack không rỗng)
    “Báo biểu thức lỗi”;
```

Stack application: Tower of Hanoi

- Tower of Hanoi game



  - Move all the disks from pole "A" to pole "B", using pole "C" as temporary storage
    - One disk at a time – Mỗi lần dịch chuyển 1 đĩa
    - Disk must rest on top of larger disk – Đĩa nhỏ phải đặt trên đĩa to



Tower of Hanoi: Ideas

- We are not writing a program to solve the puzzle automatically - Coming soon…. ☺
- Just want a simple program to allow a user to play the puzzle instead - Chương trình người chơi
  - Keep track of the discs
  - Check movement
  - Display the current state
  - Etc.
- Since we can only
  - Remove the topmost disc from a pole, then
  - Place the disc on top of other pole
- Clearly, **each pole is a stack**

Tower of Hanoi: Development

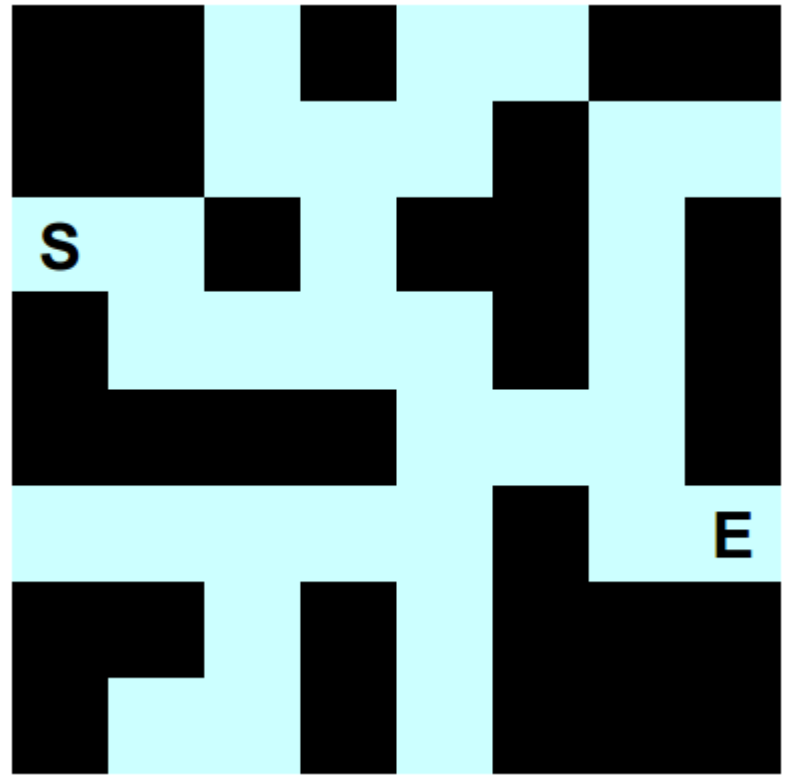
- Lets play

  http://game-game.vn/170978/

- And write your own program
  - Sử dụng stack (không phải 1 mà là 3 stack)
  - Lập trình vẽ đồ họa (2D) với java
  - Đánh dấu Written by ****.K63CIS

Stack application: Exploring a Maze

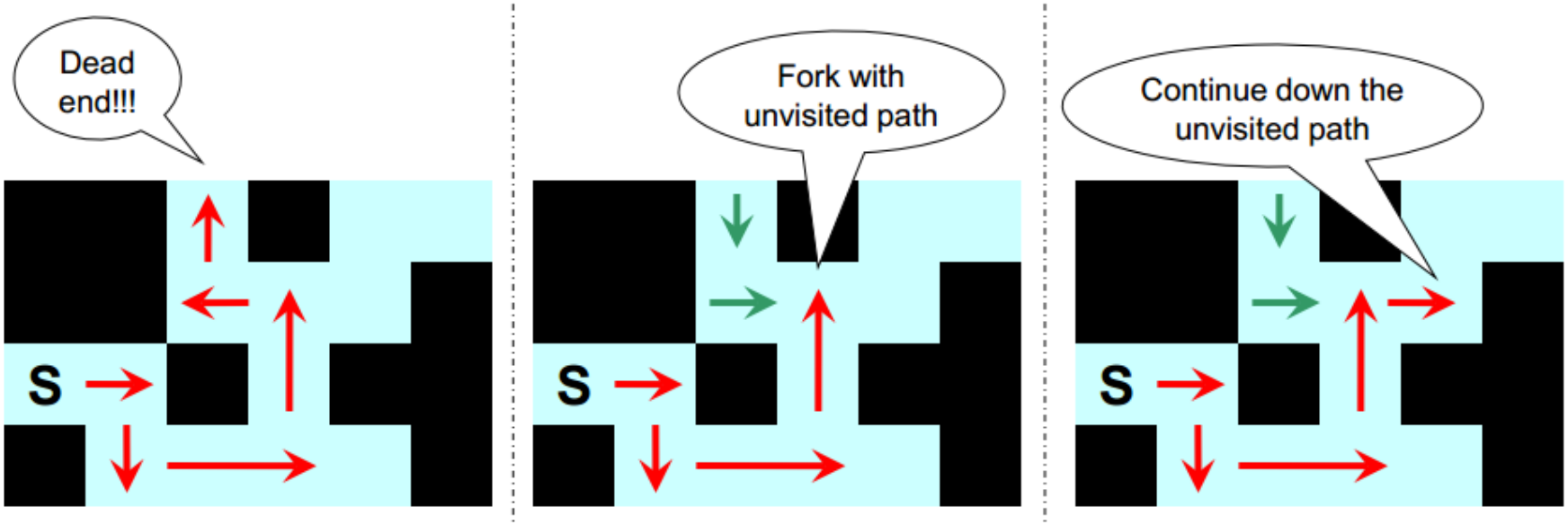
- Exploring a Maze game



- How to define an algorithm that always get you from S to E (is there a such path)?
  - What should you do when you reached a dead end?

Exploring a Maze: Iilustration

- When we reached a dead end
  - Always restart from **S** is usually not a good idea
- Instead, we retrace our steps until:
  - the most recent fork which has an unvisited path
  - take the unvisited path and continue exploration



Exploring a Maze: Ideas

- Biểu diễn mê cung bằng ma trận N x N
- Mã hóa các bước đi Direction bằng kiểu liệt kê:

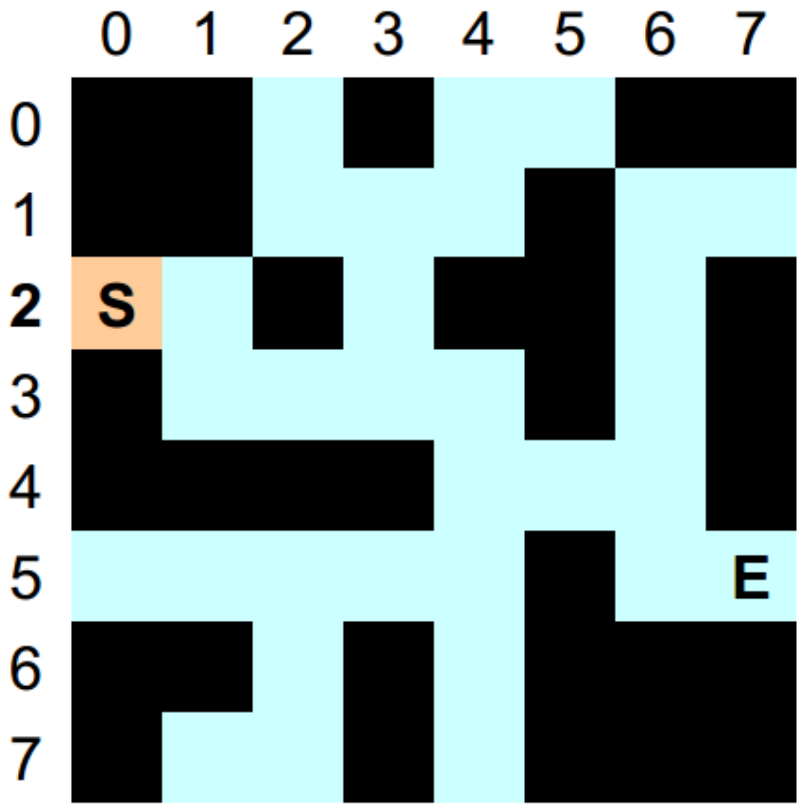
```
enum Direction {Up, Left, Down, Right, NoDir};
```

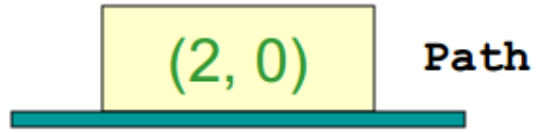
- Đối với mỗi ô của mê cung sẽ cần xác định:
  - Hướng nào chưa đi
  - Giả sử có phương thức `getUnvisitedDir()` thực hiện việc này
- Khi đến một ô có nhiều hướng đi có thể thì:
  - Đi theo thứ tự liệt kê trong `enum` ở trên
- Đường đi có thể được lưu thành các tọa độ trong một stack (a **stack of coordinates**)

Exploring a Maze: Algorithm

```
1.  Path = empty
2.  done = false //đã đến đích chưa?
3.  Path.push(coordinate of S)
4.  While (Path is not empty && not done)
    i.    CurSq = Path.top( ) //vị trí hiện thời
    ii.   NewDir = CurSq.getUnvisitedDir( )
    iii.  If (NewDir == NoDir) //ngõ cụt!
              Path.pop( ) //lùi về ô trước đó (trong stack)
    iv.    Else //nếu có một lối đi
            a) NewSq = CurSq.move( NewDir)
            b) Path.push( coordinate of NewSq)
            c) If (NewSq == E) //Yes! We reached the end!
                      done = true
```

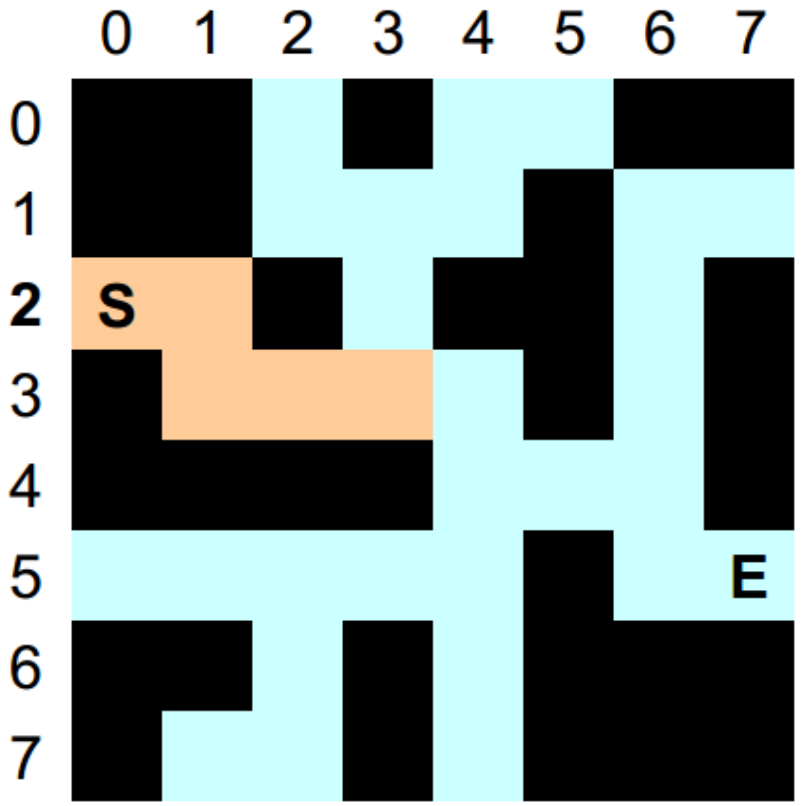
Exploring a Maze: Test run (1)

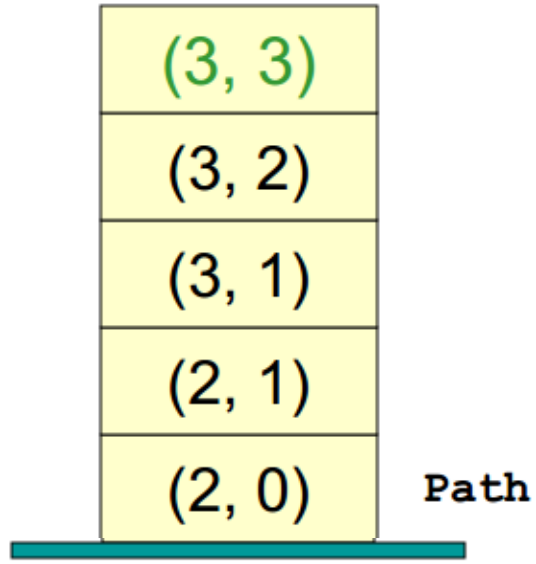




- Start at (2,0)

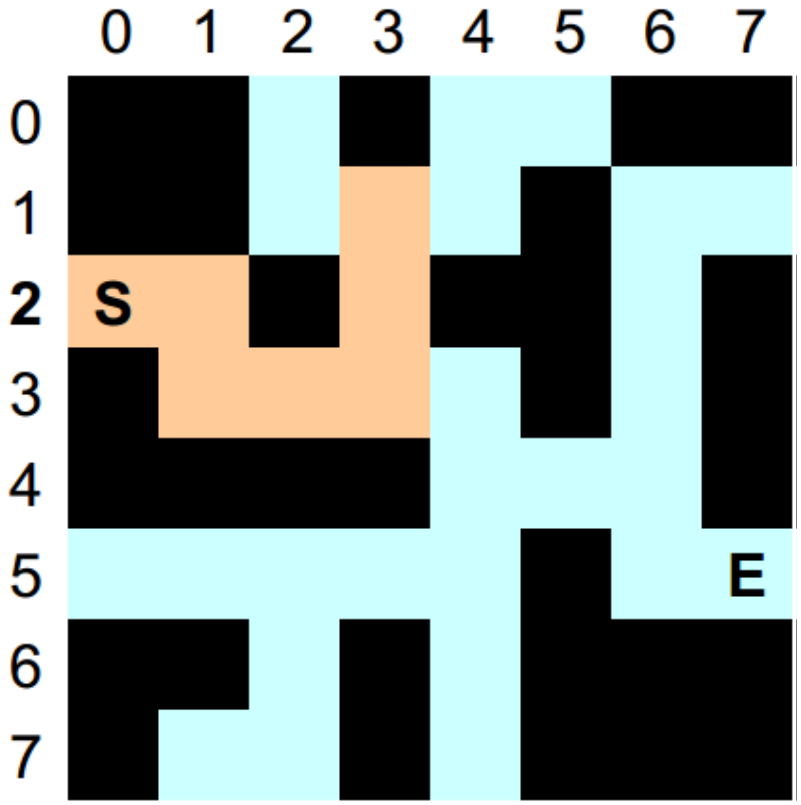
Exploring a Maze: Test run (2)

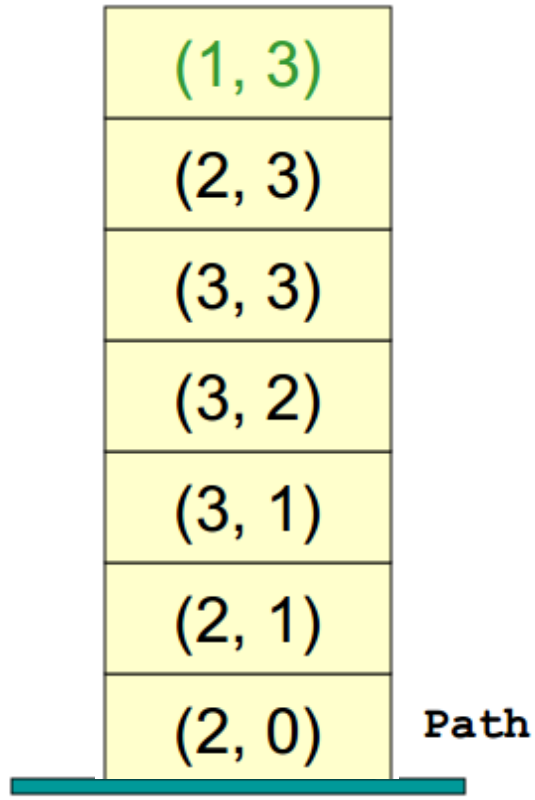




- (3,3) is the first cell with multiple exits:
  - First try go to Up

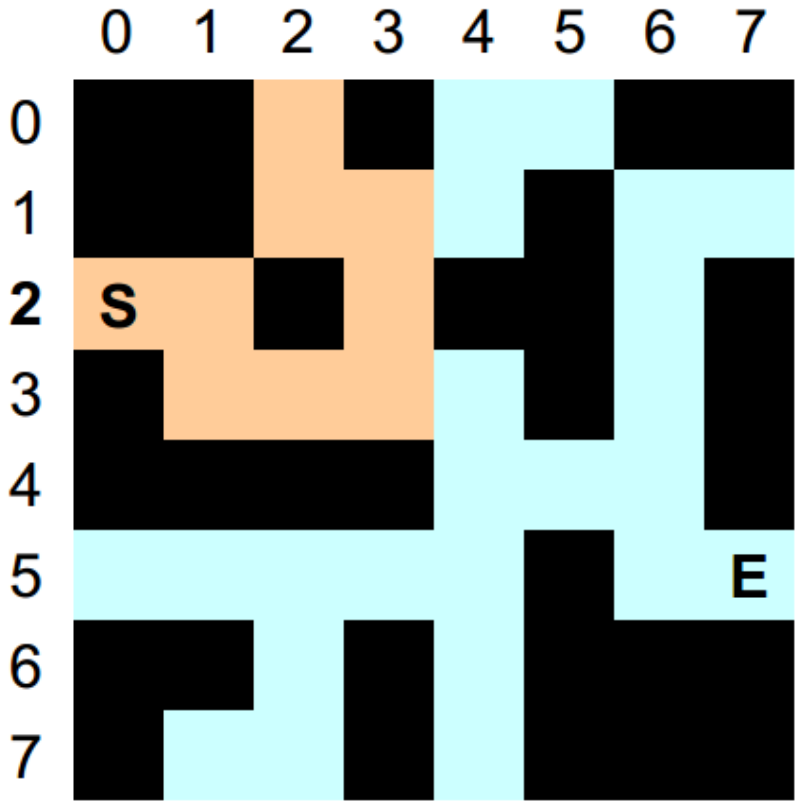
Exploring a Maze: Test run (3)

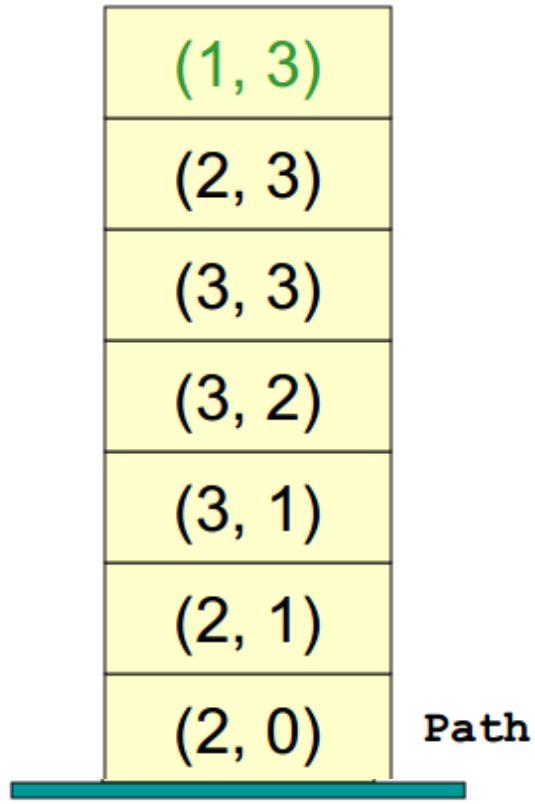




- Multiple exits at (1,3):
  - Go to Up is impossible, so go Left is the 2nd choice

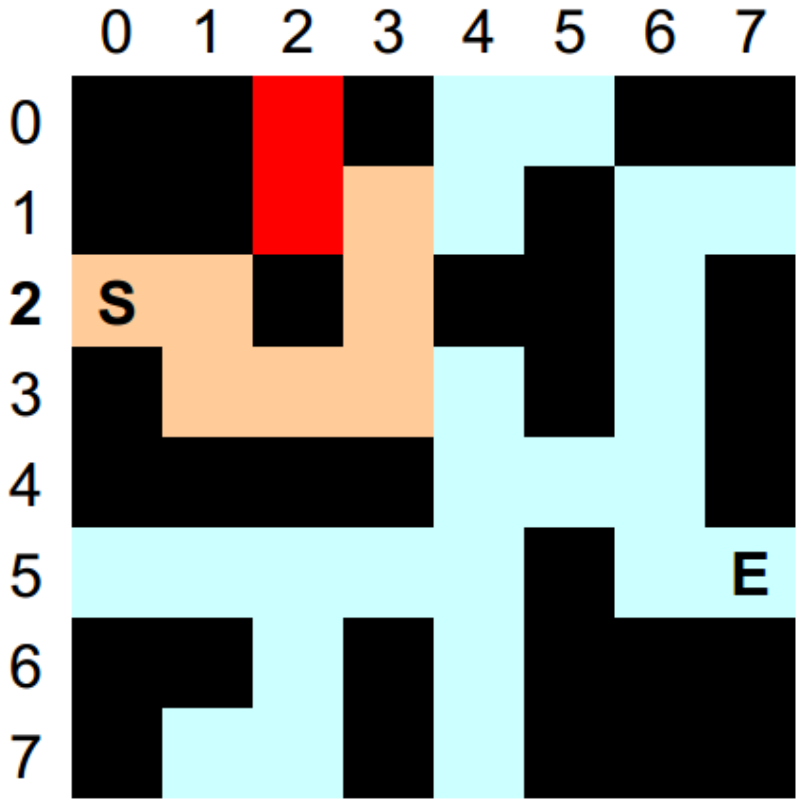
Exploring a Maze: Test run (4)

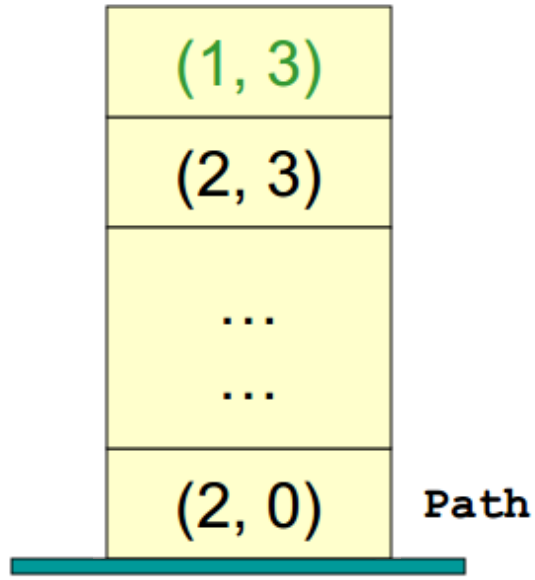




- No exits from (0,2):
  - Back trace: pop until a cell with unvisited exits

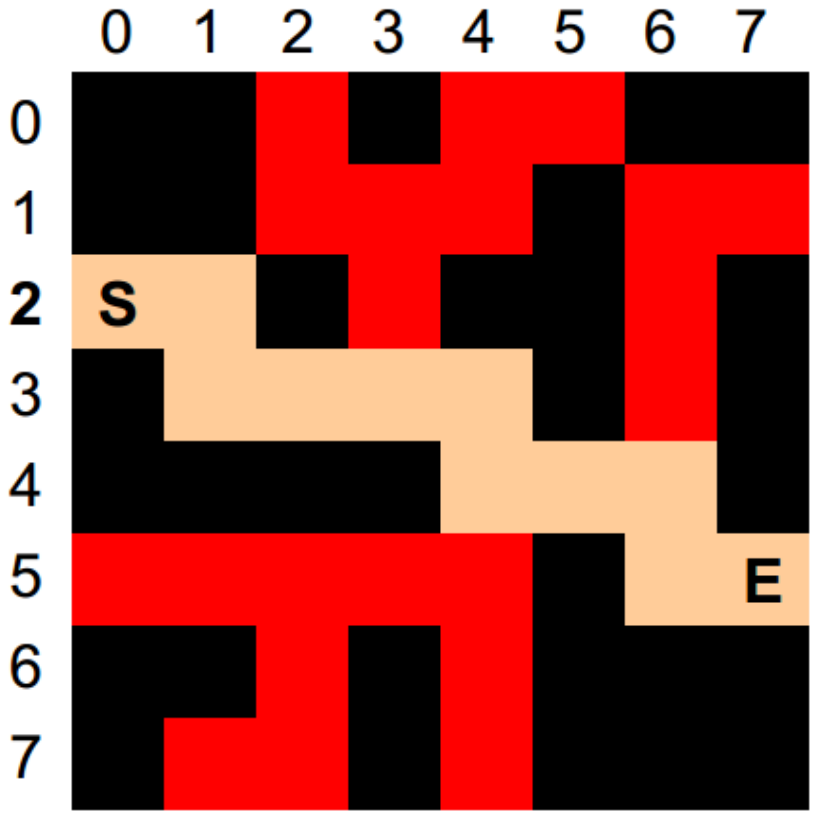
Exploring a Maze: Test run (5)

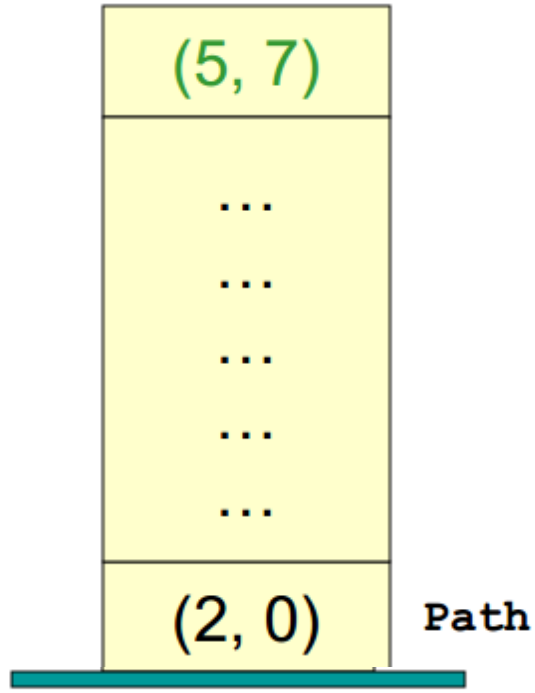




- Back to (1,3) after several pops:
  - Up, Left, Down all impossible, going Right to (1,4)

Exploring a Maze: Test run (so on)





- Stop at (5,7) at the top of the stack
  - Path found – But Poor Guy/Girl ☹

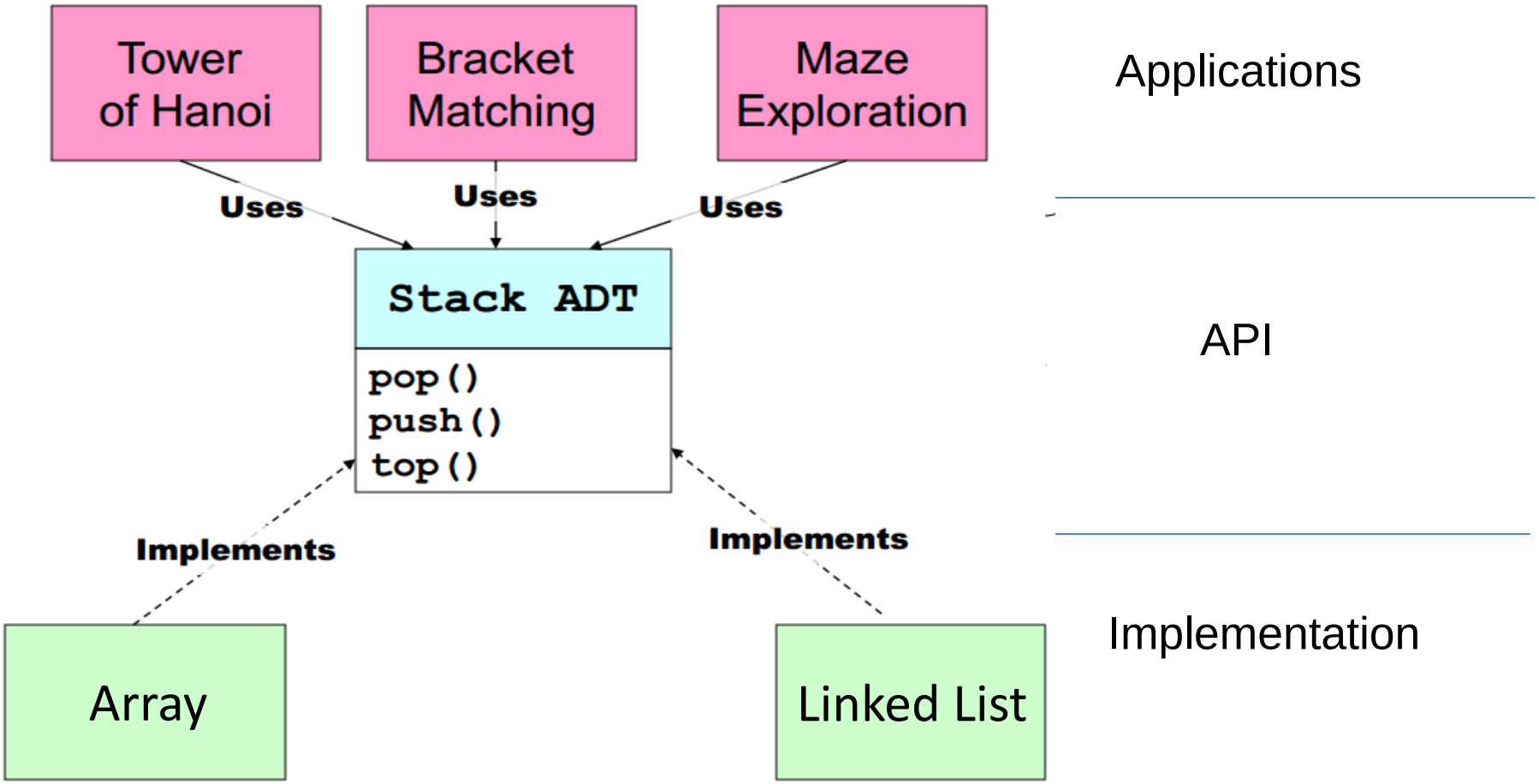
Exploring a Maze: Development

- Write your own program for exploring a maze
  - Sử dụng cấu trúc mảng 2 chiều
  - Sử dụng stack (1 stack)
  - Lập trình vẽ đồ họa (2D) với java
  - Đánh dấu - Written by ****.K63CIS

Stack ADT

Summary

Summary


Tower of Hanoi
Bracket Matching
Maze Exploration
Uses
Uses
Uses
Stack ADT
pop()
push()
top()
Implements
Implements
Array
Linked List
Applications
API
Implementation