CSE12 Discussion 4

Pooja Bhat

Discussion feedback

- https://docs.google.com/a/eng.ucsd.edu/forms/d/1rfe6mnYrINIfkWlmUnGAGY2RYTPPMSdK0YZ8Q_FMrjk/viewform?usp=send_form

Stacks

- One of the most common data structures found in computer algorithms.
- A stack is an ordered list in which all insertions and deletions are made at one end, called the top.
- LIFO

If I add elements A,B,C,D,E to a Stack, in what order will the elements be removed?

A. A B C D E
B. E D B C A
C. E D C B A
D. E A C B D

Stack using the ADP

- Create DoubleEndedLL with addFirst(), removeFirst(), addLast() and removeLast()
- Create MyStack using appropriate methods from the DoubleEndedLL
- Hint : One choice is better than the other
- Don’t forget to create the testers

QUEUE

- A queue is an ordered list in which all insertions take place at one end, the rear, while all deletions take place at the other end, the front.
- FIFO

If the elements “A”, “B”, “C” and “D” are placed in a queue and are deleted one at a time, in what order will they be removed?

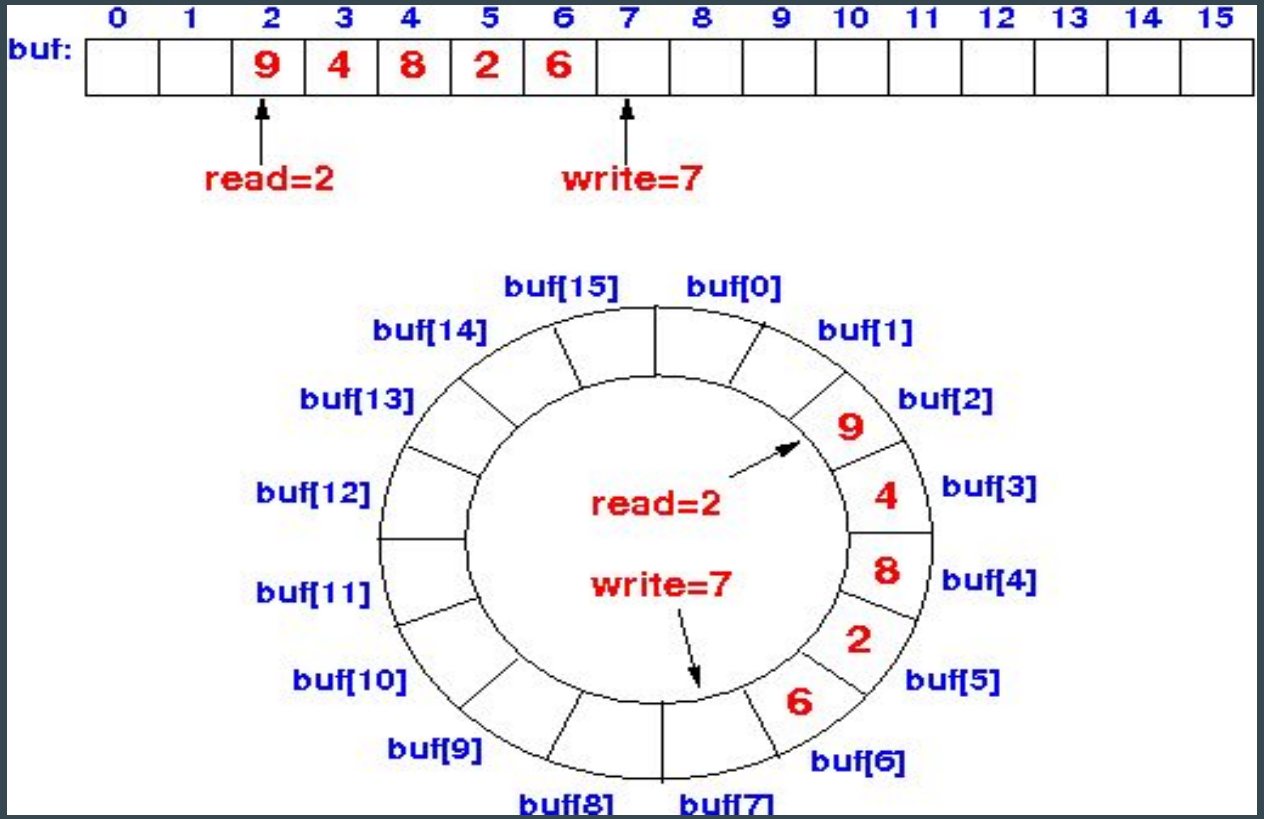
A. D C B A
B. D B A C
C. A B D C
D. A B C D

Queue using a Circular Array

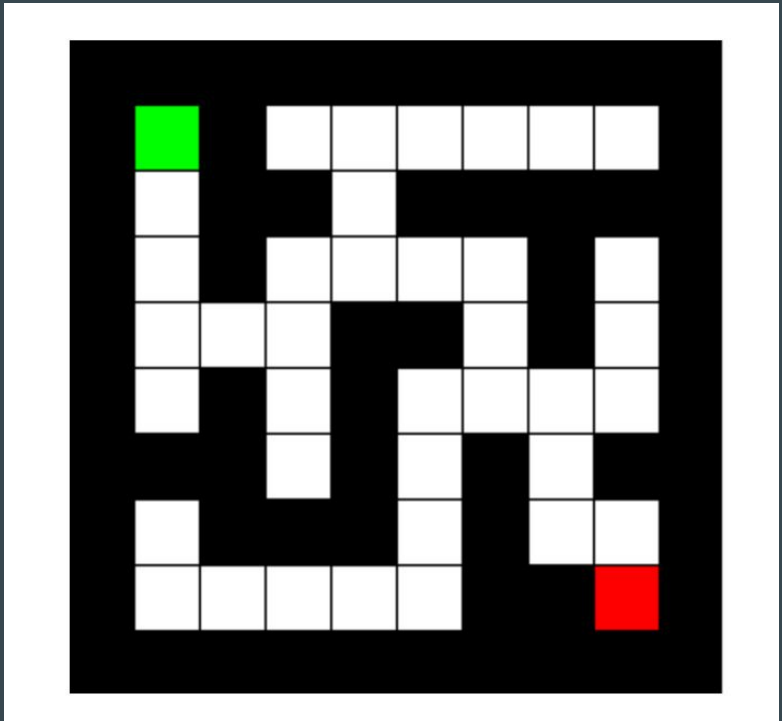
- The idea of a circular array is that the end of the array “wraps around” to the start of the array.
- Start with the initialization front = 0 and rear =0
- How do you know if the queue is empty?
  - if(front == rear)
- How do you know if the queue is full?
  - if(front == (rear+1)%size)
- The mod operator (%) is used to calculate the front and rear of the queue thus avoiding comparisons to array size.
- When to resize the queue?

If the MAX_SIZE is the size of the array used in the implementation of circular queue. How is rear manipulated while inserting an element in the queue?

A. rear = (rear % 1) + MAX_SIZE
B. rear = rear % (MAX_SIZE + 1)
C. rear = (rear + 1) % MAX_SIZE
D. rear = rear + (1 % MAX_SIZE)

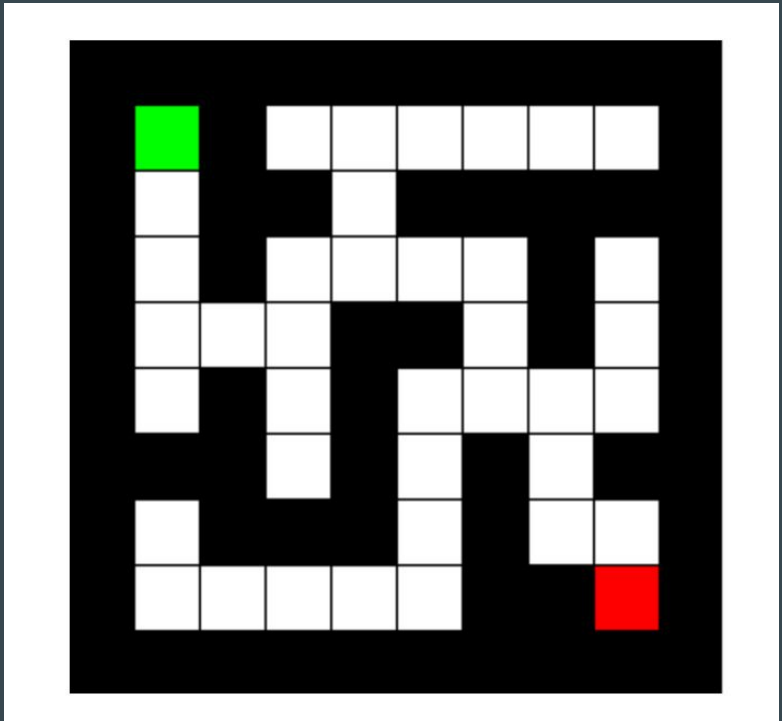

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
buf:
9 4 8 2 6
read=2
write=7
buf[15] buf[0] buf[1] buf[2] buf[3] buf[4] buf[5] buf[6] buf[7] buf[8] buf[9] buf[10] buf[11] buf[12] buf[13] buf[14]
9 4 8 2 6
read=2
write=7

Square



- Each cell in the Maze is a Square
- What do you think are the variables of a Square?
  - Row, column
  - Type of cell - Start? Exit? Wall?
- You would have getters and setters to access/modify variables
- Easy to implement. You can think of it as similar to the Node class in your DLL. Node was the unit of DLL and here Square is the unit of our Maze.

Maze



- What instance variables would a Maze have?
  - A 2D array of Squares
- Mazes are given as txt files to you.
- How to load this Maze to your 2D array?
  - Don't worry as that code is already provided to you :)
- Implement other methods of the Maze such as:
  - getNeighbors(Square sq) : returns all valid neighbors. For example, neighbors of (0,7) are?
  - Getters and Setters
  - toString() is already provided

What are the valid neighbors of Square (2,1) ?

	0	1	2	3
0				E
1			#	#
2			#	
3	S	#		#

A. (1, 1), (2, 2), (3, 1), (2,0)
B. (1, 1), (2, 2), (3, 1)
C. (1, 1), (2, 0)
D. (2, 0), (3, 1)

MazeSolver algorithm - Stack

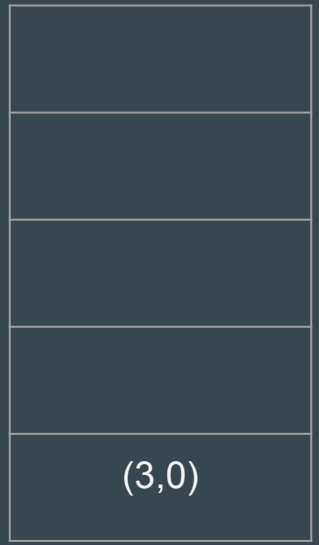
	0	1	2	3
0				E
1			#	#
2			#	
3	S	#		#

- Initialize worklist = Start
- While worklist is not empty, do
  - Get next location from worklist.
  - Is it an Exit?
    - If yes, Yayy we found the escape!!
  - If not, then explore it as follows:
    - Find its valid neighbors in the order Up, Right, Down, Left
    - Add them to the worklist to be explored later (if they have not been already added)
  - Mark it as visited so that it is not explored again
- If worklist is empty and no escape has been found, then there is no escape from the maze :(

MazeSolver algorithm - Stack

Initialize

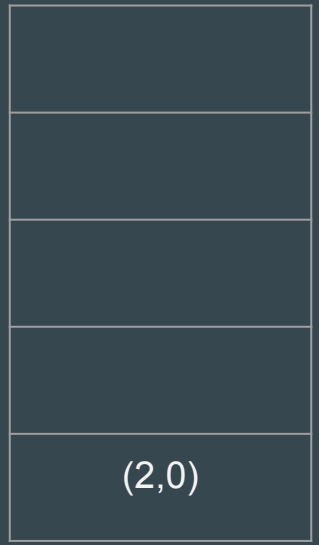
	0	1	2	3
0				E
1			#	#
2			#	
3	S	#		#



MazeSolver algorithm - Stack

Current = (3,0)

	0	1	2	3
0				E
1			#	#
2			#	
3	S	#		#



MazeSolver algorithm - Stack

Current = (2,0)

	0	1	2	3
0				E
1			#	#
2	.		#	
3	S	#		#

(2,1)
(1,0)

MazeSolver algorithm - Stack

Current = (2,1)

	0	1	2	3
0				E
1			#	#
2	.	.	#	
3	S	#		#

(1,1)
(1,0)

MazeSolver algorithm - Stack

Current = (1,1)

	0	1	2	3
0				E
1		.	#	#
2	.	.	#	
3	S	#		#

(0,1)
(1,0)

MazeSolver algorithm - Stack

Current = (0,1)

	0	1	2	3
0		.		E
1		.	#	#
2	.	.	#	
3	S	#		#

(0,0)
(0,2)
(1,0)

MazeSolver algorithm - Stack

Current = (0,0)

	0	1	2	3
0	.	.		E
1		.	#	#
2	.	.	#	
3	S	#		#

(0,2)
(1,0)

MazeSolver algorithm - Stack

Current = (0,2)

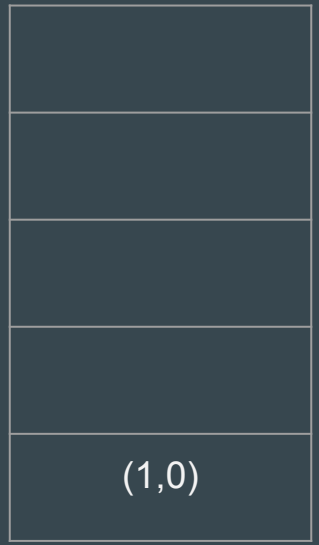
	0	1	2	3
0	.	.	.	E
1		.	#	#
2	.	.	#	
3	S	#		#

(0,3)
(1,0)

MazeSolver algorithm - Stack

Current = (0,3)

	0	1	2	3
0	.	.	.	E
1		.	#	#
2	.	.	#	
3	S	#		#



MazeSolver algorithm - Stack

	0	1	2	3
0	.	.	.	E
1		.	#	#
2	.	.	#	
3	S	#		#

Tracing Path.

- Start from E and keep track of who added it to the worklist

MazeSolver algorithm - Stack

	0	1	2	3
0	.	x	x	E
1	o	x	#	#
2	x	x	#	
3	S	#		#

Tracing Path.

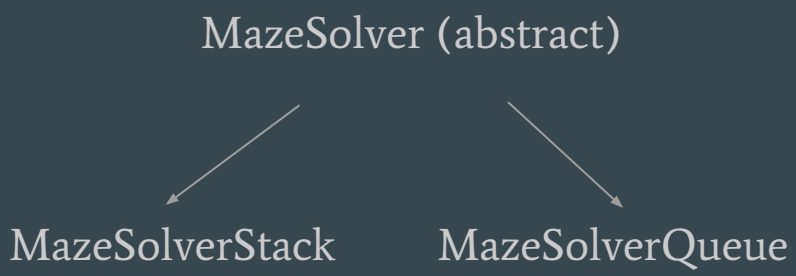
- Start from E and keep track of who added it to the worklist
- This is where it is useful to have each Square keep track of a **previous** Square that added it to the worklist

An abstract class can have both abstract as well as non abstract methods

A.. True

B. False

MazeSolver



Create an abstract class MazeSolver that will implement the above algorithm, with a general worklist.

Its abstract methods will be implemented differently depending on whether the worklist is a stack or a queue.

You will also have non-abstract methods that implement the algorithm for solving the maze in the MazeSolver class.

Stack and Queue Applications Midterm/Interview Prep

Balanced Parentheses

Given an expression string, write a program to examine whether the pairs and the orders of “{“,”}”,”(“,”)”,”[“,”]” are correct in expression.

For example, the program should print true for exp = “[()]{}{[()()]()}” and false for exp = “[(])”.

What data structure do you think will be most useful?

Rotten Oranges - Extra Question

Given a matrix of dimension m*n where each cell in the matrix can have values 0, 1 or 2 which has the following meaning:

0: Empty cell
1: Cells have fresh oranges
2: Cells have rotten oranges

So we have to determine what is the minimum time required so that all the oranges become rotten. A rotten orange at index [i,j] can rot other fresh orange at indexes [i-1,j], [i+1,j], [i,j-1], [i,j+1] (up, down, left and right) in one time frame. If it is impossible to rot every orange then simply return -1.
.....continued in next slide

Rotten Oranges

Input: arr[3][5] = { {2, 1, 0, 2, 1},
{1, 0, 1, 2, 1},
{1, 0, 0, 2, 1}};

Output:

All oranges can become rotten in 2 time frames.

Input: arr[ ][ ] = { {2, 1, 0, 2, 1},
{0, 0, 1, 2, 1},
{1, 0, 0, 2, 1}};

Output:

All oranges cannot be rotten.