CSE 250 - Data Structures: Stacks and Queues

Andrew Hughes

SUNY at Buffalo
Computer Science and Engineering

Fall 2019

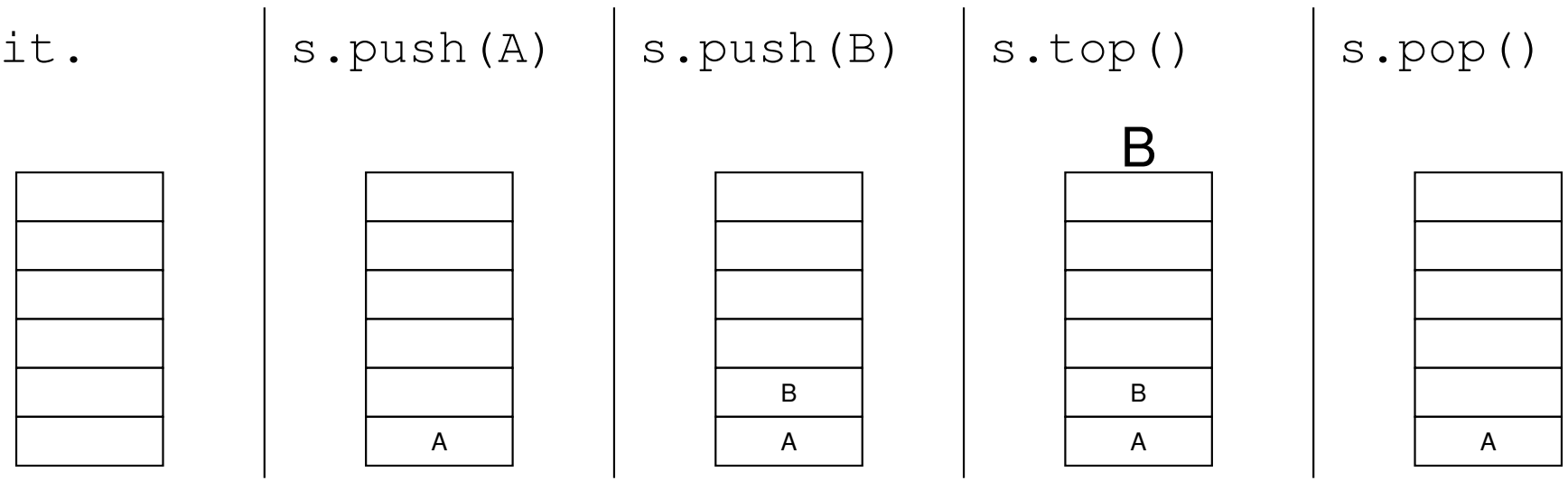
Outline

What is a Stack?

Idea:

Store of objects/values stacked on top of one another.

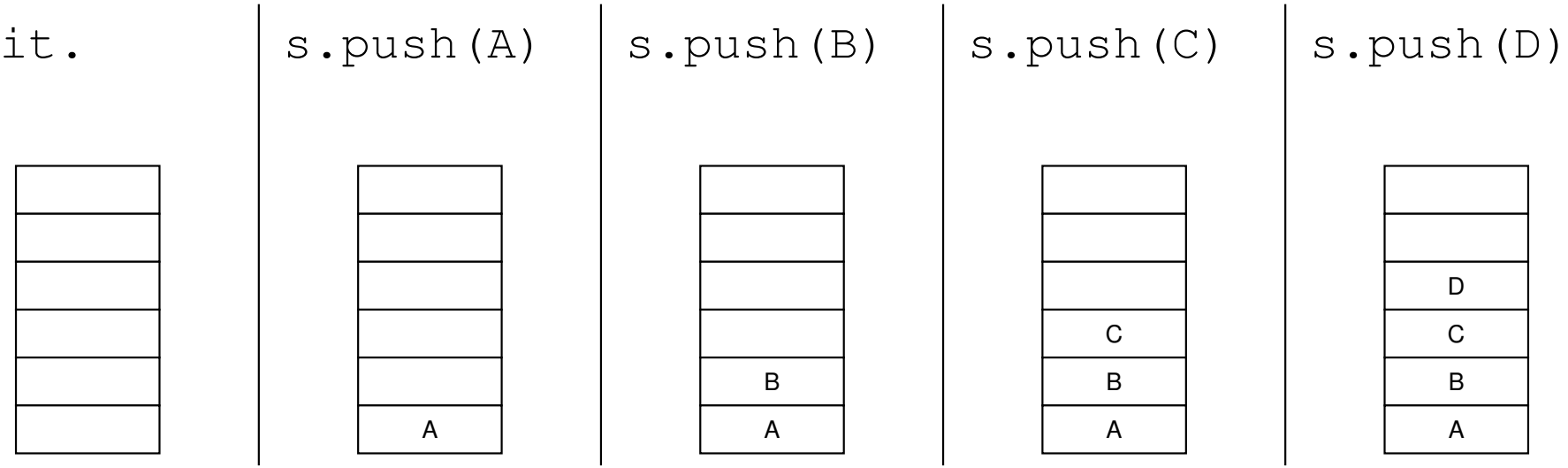
```
val s = new Stack[A]
```



push

- We `push` items onto the stack.

```
val s = new Stack[A]
```

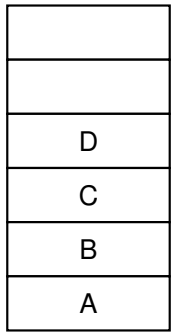


top

- We can access the element on `top` of the stack.

```
s.top()
```

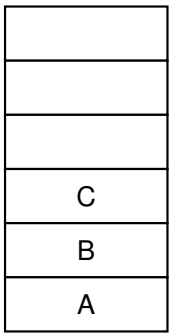
D



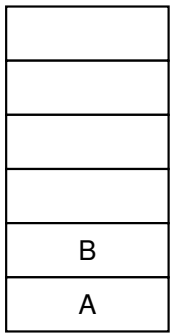
pop

- We can `pop` objects off of the stack.

`s.pop()`



`s.pop()`



`s.pop()`

`s.pop()`

`s.pop()`

How can we implement one?

```
trait Stack[A]  {
  def push(value: A): Unit
  def top(): A
  def pop(): Unit
  def empty(): Boolean
}
```

Idea:

Implement a stack using `ListADT[A]`.

- Wrap functionality of: `val data:ListADT[A]`

Stack Operations

`push`: insert to end of list.

```
def push(value: A): Unit = {
    data.insert(data.length,value)
}
```

`top`: retrieve last element of list.

```
def top: A = {
    data(data.length-1)
}
```

`pop`: remove last element of list.

```
def pop: Unit = {
    data.remove(data.length-1)
}
```

Stack Operations

Stack Function	Runtime
`push()`	??
`top()`	??
`pop()`	??

Where are Stacks used?

Stacks are used in many places:

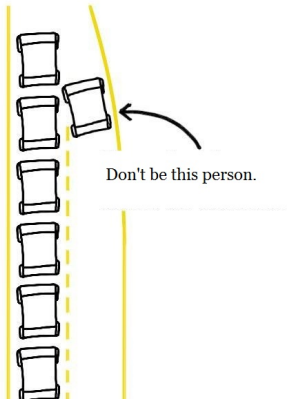
- Function calls are stored in a “call stack” by operating system.
- Can be used to parse expressions (anything “Context Free”).
- Backtracking.
- Reverse sequences.

Outline

What is a queue?

For the rest of the world...

"Queuing" is the same as "lining up."



Source [1]

Allows the first person in to be the first person out.

- First in, first out (FIFO).

Contrast with stack: the last person in is the last person out.

- Last in, first out (LIFO).

Queue Functions

Consider the process of queuing at a store.

- A person enters the back of the queue. (`enqueue(item)`)
- A person leaves the front of the queue. (`dequeue()`)
- A clerk can observe who is next. (`front()`)

Stack vs Queue Functions Delegated to Container

```
trait Queue[A]  {
  def enqueue(value: A): Unit
  def front(): A
  def dequeue(): Unit
  def empty(): Boolean
}
```

Stack:

- `push(item)`:
  Insert to end of list.
- `pop()`:
  Remove from end of list.
- `top()`:
  Retreive end of list.

Queue:

- `enqueue(item)`:
  Insert to end of list.
- `dequeue()`:
  Remove from front of list.
- `front()`:
  Retreive front of list.

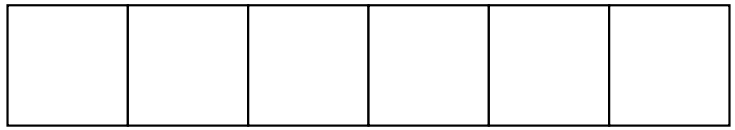
Stack vs Queue Behavior

```
push(1);push(2);push(3);push(4)
```

Stack:

```
top(); pop();top(); pop();top(); pop(); top(); pop()
```

order: 4, 3, 2, 1

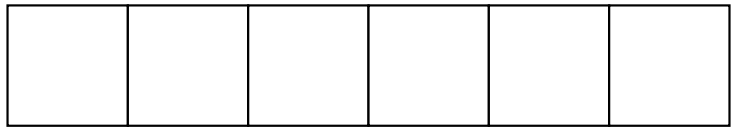


```
enqueue(1);enqueue(2);enqueue(3);enqueue(4)
```

Queue:

```
front(); dequeue();front(); dequeue();front();
dequeue();front(); dequeue()
```

order: 1, 2, 3, 4



Queue Operations

Our runtimes:

Queue Function	Runtime
`enqueue()`	??
`front()`	??
`dequeue()`	??

Outline

Matching punctuation

Problem

How can we check if code has balanced parentheses/braces?

- Balanced parens/braces means:
  - Every opening symbol has a closing symbol proceeding it.
  - Pairs are properly nested.
    - Open and close are not separated by an unmatched pair.
    - Can have “{ ( ) ({ }) }”.
    - Cannot have “{ (})”, “{ ( )” or “( ))”.

- Push opening symbols to stack.
- Check closing symbols with top of stack and if matching, pop.

```
if(iterator.hasNext) {
    println(s"Next: ${iterator.next}'')
}
else {
     println("At the end!")
}
```

Network Packet Queue

Problem

How can we process data packets through a computer network?

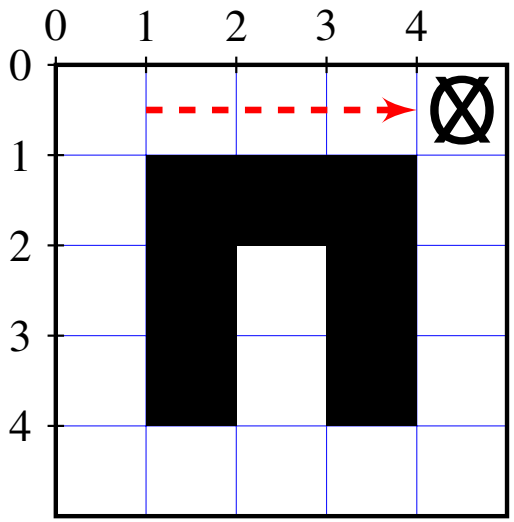
- Push data packets to queue in order received.
- Process front and pop.

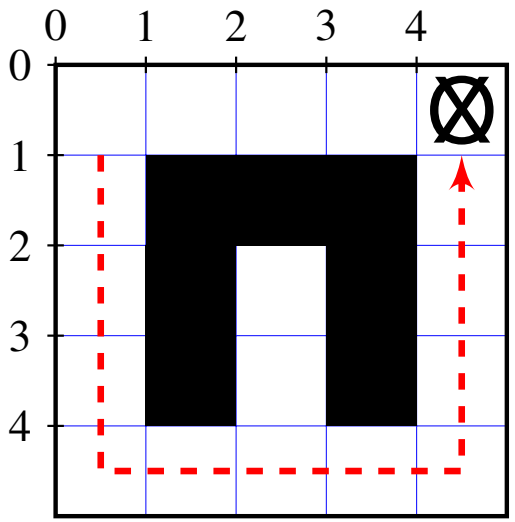
Network switches and access points are built to be fast.

- Avoid delay by setting max queue size.
  - Drop incoming packets that don't fit.
- If data doesn't reach destination:
  - TCP – keep resending until return receipt received.
  - UDP – oh well.

Navigating a maze

- Suppose we are the circle O and want to navigate to the X.
  - Generally want the shortest path to our goal.
  - There may be multiple paths to our goal.

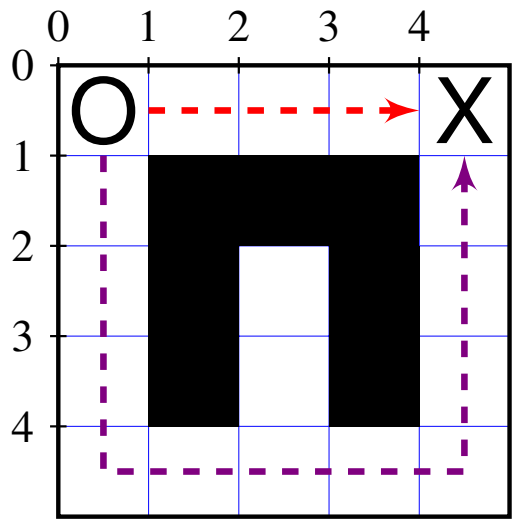


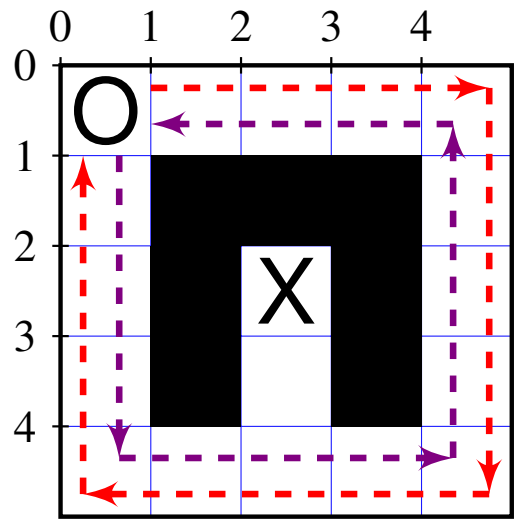


Navigating a maze

How do we navigate the maze?

- Choose one direction to go first?
- **Right, then down, then left, then up?**
- **Down, then right, then up, then left?**





Navigating a maze

Problem

Find the shortest path from current position to destination.

Minimize the following:

$$steps(pos, dest) = 1 + \min \begin{Bmatrix} steps(moveR(pos), dest), \\ steps(moveD(pos), dest), \\ steps(moveL(pos), dest), \\ steps(moveU(pos), dest). \end{Bmatrix}$$

Navigating a maze

How can this be done in code?

- Create a map of maze in global space (or pass reference).
- `steps(pos,dest)`:
  1. Mark `pos` visited.
  2. If `pos == dest`:
     - ★ Unmark `pos`.
     - ★ Return 0.
  3. Else, compute the minimum of the following calls:
     - ★ If right is open/unvisited: `steps(moveR(pos),dest)`
     - ★ If down is open/unvisited: `steps(moveD(pos),dest)`
     - ★ If left is open/unvisited: `steps(moveL(pos),dest)`
     - ★ If up is open/unvisited: `steps(moveU(pos),dest)`
  4. Unmark `pos`.
  5. Return 1 + min from recursive calls.

Navigating a maze

Idea

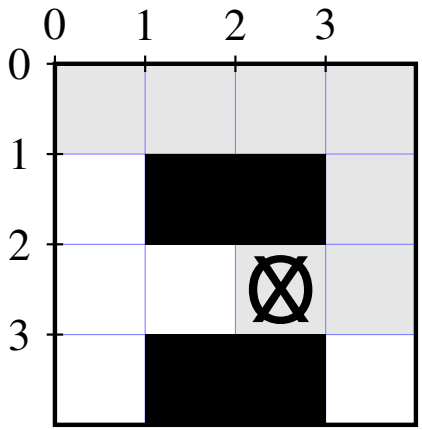
Use stack/queue to monitor nodes to visit.

- Stack Idea:
  - Store function frame info on stack.
  - When we hit a return, backtrack.

Navigating a maze with a stack

Push locations visited with direction last explored.

- Recall visit order: right, down, left, up.
- Height of stack at end will give us distance.



(2,3), L
(1,3), D
(0,3), D
(0,2), R
(0,1), R
(0,0), R

Stack

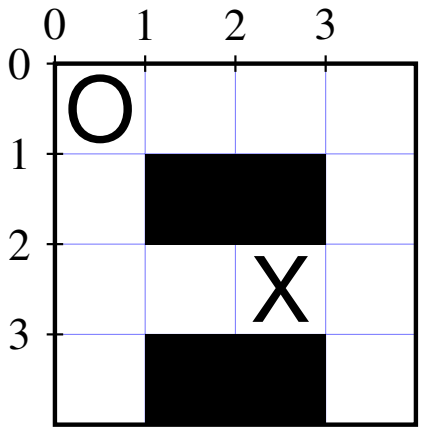
Shortest Path: 6

- Continue until stack is cleared.
  - Explore all paths.

Navigating a maze with a stack

Push locations visited with direction last explored.

- Recall visit order: right, down, left, up.
- Height of stack at end will give us distance.



Stack
Shortest Path: 4

- Done. Shortest path is 4.

Navigating a maze

Idea

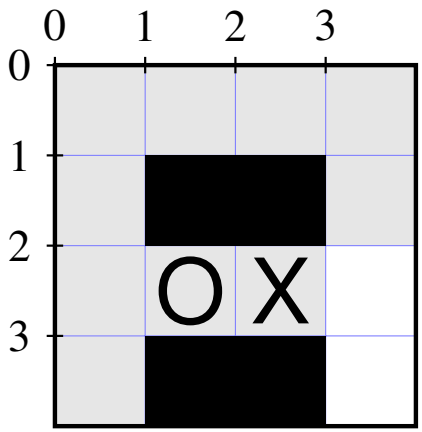
Use stack/queue to monitor nodes to visit.

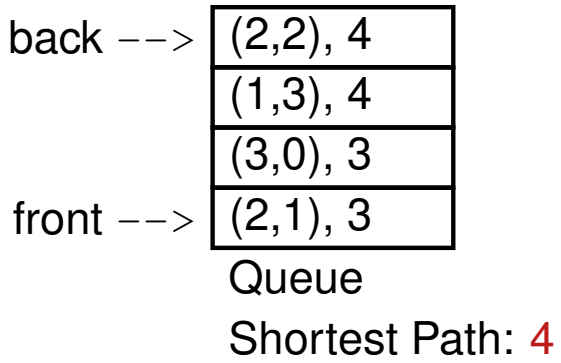
- Queue Idea:
  - Go down every path simultaneously, radially.
  - Stop when destination is reached.

Navigating a maze with a queue

Push all reachable locations, paired with distance, and mark visited.

- Recall visit order: right, down, left, up.
- Distance passed along as we queue.





Shortest Path: 4

- We go out in every direction one step at a time.
  - Guarantees shortest path.

Bibliography

 “funny-car-photos-simple-fact-infographic-comic-how-it-is.jpg.” [Online; accessed October 15, 2019].