Lecture 08: Basic Data Structures BT 3051 – Data Structures and Algorithms for Biology

Karthik Raman

Department of Biotechnology Bhupat and Jyoti Mehta School of Biosciences Indian Institute of Technology Madras

August 31, 2018

Stacks

- Stacks and queues are more abstract entities than arrays and many other data storage structures
- They're defined primarily by their interface the permissible operations that can be carried out on them
- The underlying mechanism used to implement them is typically not visible to their users
- The underlying mechanism for a stack can be an array or a linked list

- Stacks and queues are more abstract entities than arrays and many other data storage structures
- ► They're defined primarily by their interface the permissible operations that can be carried out on them
- The underlying mechanism used to implement them is typically not visible to their users
- The underlying mechanism for a stack can be an array or a linked list

- Stacks and queues are more abstract entities than arrays and many other data storage structures
- ► They're defined primarily by their interface the permissible operations that can be carried out on them
- The underlying mechanism used to implement them is typically not visible to their users
- The underlying mechanism for a stack can be an array or a linked list

- Stacks and queues are more abstract entities than arrays and many other data storage structures
- ► They're defined primarily by their interface the permissible operations that can be carried out on them
- The underlying mechanism used to implement them is typically not visible to their users
- ► The underlying mechanism for a stack can be an array or a linked list

Stack

► Can be implemented using Arrays or Linked Lists

Operations

- create()
- ▶ delete()
- ▶ isEmpty()
- push()
- **▶** pop()
- ▶ top()

Delimiter Matching using a Stack

```
from ArrayStack import *
def is_matched(expr):
    '''Return True if all delimiters are properly match;
                                  False otherwise.'''
    lefty = '({[' # opening delimiters
    righty = ')}]' # respective closing delims
    S = ArrayStack()
    for c in expr:
        if c in lefty:
            S.push(c) # push left delimiter on stack
        elif c in righty:
            if S.is_empty():
                return False # nothing to match with
            if righty.index(c) != lefty.index(S.pop()):
                return False # mismatched
    return S.is_empty() # were all symbols matched?
```

Reversing a String using a Stack

```
from ArrayStack import *
def string_reverser(s):
    S = ArrayStack()
    for c in s:
        S.push(c)
   r = []
    while not S.is_empty():
        r.append(S.pop())
    return ''.join(r)
if __name__ == '__main__':
   print(string_reverser('abcdef'))
```

- Push operands onto the operand stack
- Push operators onto the operator stack
- ► Ignore left parentheses
- On encountering a right parenthesis, pop an operator, pop the requisite number of operands, and push onto the operand stack the result of applying that operator to those operands

- Push operands onto the operand stack
- Push operators onto the operator stack
- ► Ignore left parentheses
- On encountering a right parenthesis, pop an operator, pop the requisite number of operands, and push onto the operand stack the result of applying that operator to those operands

- Push operands onto the operand stack
- Push operators onto the operator stack
- Ignore left parentheses
- On encountering a right parenthesis, pop an operator, pop the requisite number of operands, and push onto the operand stack the result of applying that operator to those operands

- Push operands onto the operand stack
- Push operators onto the operator stack
- Ignore left parentheses
- On encountering a right parenthesis, pop an operator, pop the requisite number of operands, and push onto the operand stack the result of applying that operator to those operands

Using Stacks to Replace Recursion

Stacks underlie recursion in computers!

```
from ArrayStack import *
def factorial_stack(n):
    S = ArrayStack()
    while n>1:
        S.push(n)
        n -= 1
    result = 1
    while not S.is_empty():
        result *= S.pop()
    return result
if __name__ == '__main__':
    print(factorial_stack(10))
```

Other Stack Scenarios

- 'Back button' on your browser
- ► Undo stack
- **.**.



Queue

Queues can be implemented using an array or linked list

Operations

- create()
- ► delete()
- ▶ isEmpty()
- ▶ length()
- ▶ enqueue()
- ▶ dequeue()
- ▶ front()

Array-based Queue implementation

- ► How will you implement a queue using an array?
- ► Can the same stack idea work?
- Self-assessment Exercise: Implement ArrayQueue class

Array-based Queue implementation

- ► How will you implement a queue using an array?
- ► Can the same stack idea work?
- Self-assessment Exercise: Implement ArrayQueue class

Array-based Queue implementation

- ► How will you implement a queue using an array?
- ► Can the same stack idea work?
- Self-assessment Exercise: Implement ArrayQueue class