Introduction to Programming and Data Structures

Stack

Malay Bhattacharyya

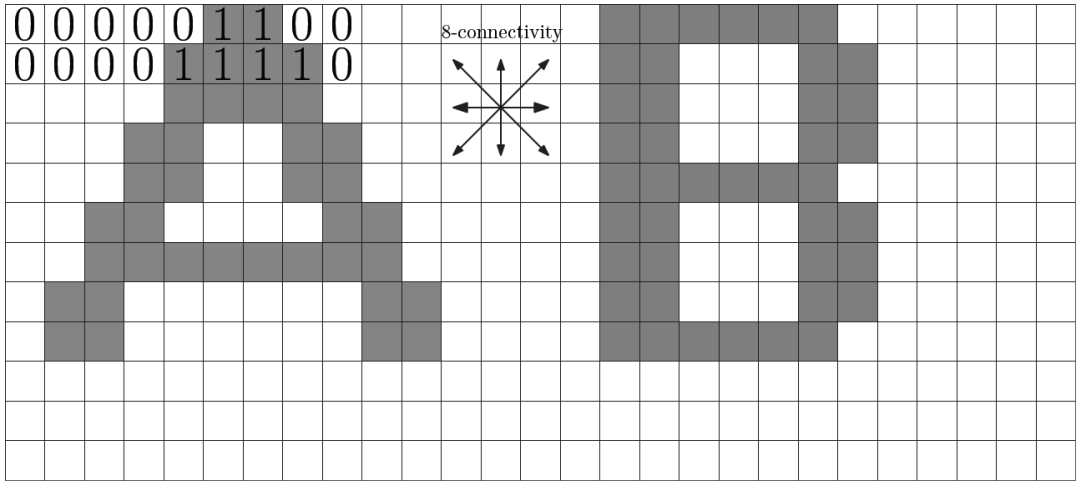
Associate Professor

MIU, CAIML, TIH
Indian Statistical Institute, Kolkata

October, 2023

Motivating example

Problem: Find the *connected components* in an image.



Data structure: Stack (LIFO)

Operations:

- `PUSH`: Insert at the top
- `POP`: Remove and return element from the top
- `DISPLAY`: Show the elements
- `LENGTH` or `HEIGHT`
- `ISEMPTY`, `ISFULL`

Initialization

Considering the stack size as static:

```
StackDS = []
SizeStackDS = 10
```

Push operation

```
def StackPUSH(Data, StackDS):
    if len(StackDS) < SizeStackDS:
        StackDS.append(Data)
    else:
        print("Stack overflow!!!")
```

Pop operation

```
def StackPOP(StackDS):
    if len(StackDS) > 0:
        Data = StackDS.pop()
        return Data
    else:
        print("Stack underflow!!!")
```

Note: If you do not wish to receive the popped element, exclude return statement.

Displaying the elements

```
def StackDISPLAY(StackDS):
    print("The elements in the stack are: ")
    for Element in StackDS:
        print(Element)
```

Note: You can directly write `print(StackDS)`.

Infix, Prefix and Postfix expression

Infix expression:

Operand1 Operator Operand2

Prefix expression:

Operator Operand1 Operand2

Postfix expression:

Operand1 Operand2 Operator

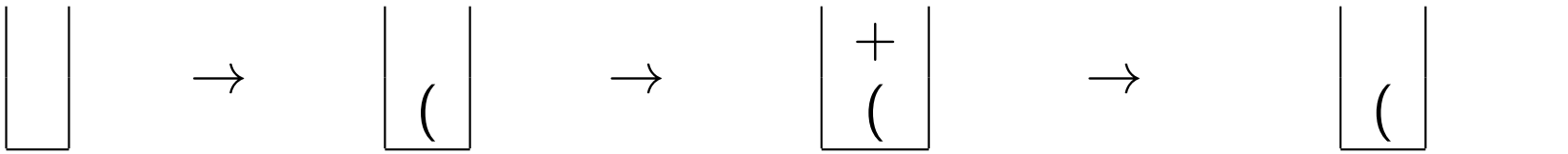
Converting Infix to Postfix expression

1. Create an empty stack and an empty postfix expression.
2. Iterate through the infix expression from left to right and append operands to the postfix expression.
3. If an operator is encountered, pop operators from the stack and append them to the postfix expression until an operator with lower or equal precedence is found.
4. Push the current operator onto the stack.
5. If a left parenthesis is encountered, push it onto the stack.
6. If a right parenthesis is encountered, pop operators from the stack and append them to the postfix expression until a left parenthesis is found.
7. Pop any remaining operators from the stack and append them to the postfix expression.

Converting Infix to Postfix expression

Input:

```
(a + b) * c / d
```



Output:

```
a b + c d / *
```

Depth-first Search (DFS)

We can perform depth-first search on a graph given in the form of adjacency list in a file. The DFS algorithm works as follows:

i) Keep any one of the graph's vertices on top of a stack.

ii) Pop out the top data item from the stack and add it to the Visited list.

iii) Create a list of that vertex's adjacent nodes. Add the ones which are not in the Visited list to the top of stack.

iv) Repeat steps (ii)-(iii) until the stack is empty.

Implementing the DFS

```
# Adjacency list defined as a dictionary
Graph = {
    '0' : ['1','2'], '1' : ['3', '4'], '2' : ['5'],
    '3' : [], '4' : ['5'], '5' : []
}
Visited = [] # Array to keep track of visited vertices
def DFS(Visited, Graph, Vertex):
    if Vertex not in Visited:
        Visited.append(Vertex)
        print(Visited[len(Visited) - 1]) # Top of stack
        for Adjacent in Graph[Vertex]:
            DFS(Visited, Graph, Adjacent)
DFS(Visited, Graph, '0') # Function call with vertex '0'
```

Problems

1. Given a list of numbers (provided as command line arguments), write a Python program to compute the nearest larger value for the number at position i (nearness is measured in terms of the difference in array indices). For example, in the array $[1, 4, 3, 2, 5, 7]$, the nearest larger value for 4 is 5. Note: You are expected to design an $O(n)$ time algorithm.
2. Write a Python program to convert an input algebraic expression into its postfix form.
3. Write a Python program to implement two different stacks within a single array such that both of them grows toward each other being in the same array.