Revision - Stacks

Introduction

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. In simpler terms, the last element that was added to the stack will be the first one to be removed. A stack has two main operations, namely push and pop. Push is used to add an element to the top of the stack, while pop is used to remove the topmost element.

Basic Functions of Stack

Push

This function is used to add an element to the top of the stack. It takes one argument, which is the element to be added. The time complexity of push operation is O(1).

Pop

This function is used to remove the topmost element from the stack. It does not take any argument. The time complexity of pop operation is also O(1).

Peek

This function is used to get the value of the topmost element without removing it from the stack. It does not take any argument. The time complexity of peek operation is O(1).

Implementation

```
int stack[MAX_SIZE];
int top;

void push(int element) {
    if (top < MAX_SIZE - 1) {
        top++;
        stack[top] = element;
    } else {
        cout << "Stack Overflow" << endl;
    }
}

int pop() {
    if (top >= 0) {
        int element = stack[top];
        top--;
        return element;
    } else {
        cout << "Stack Underflow" << endl;
    }
}

bool isEmpty() {
    return top < 0;
}

int peek() {
    if (!isEmpty()) {
        return stack[top];
    } else {
        cout << "Stack is empty" << endl;
    }
}
```

Parenthesis Check

One of the most common problems that can be solved using a stack is to check if the given string of parentheses is balanced or not. A string of parentheses is considered balanced if every opening parenthesis has a corresponding closing parenthesis and they appear in the correct order. For example, "((()))" and "()()()" are balanced strings of parentheses, while "(()" and ")(" are not.
The algorithm to check if a given string of parentheses is balanced using a stack is as follows:

- Create an empty stack.
- Traverse the string of parentheses from left to right.
- If the current character is an opening parenthesis, push it to the stack.
- If the current character is a closing parenthesis, pop the topmost element from the stack and check if it is the corresponding opening parenthesis. If not, the string of parentheses is not balanced.

- If the stack is empty at the end of the traversal, the string of parentheses is balanced.

Code

```
is_balanced(string):
    stack = empty stack
    for each character c in string:
        if c is an opening parenthesis:
            push(stack, c)
        else if c is a closing parenthesis:
            if stack is empty or peek(stack) is not the corresponding opening pa
                return false
            else:
                pop(stack)
    return true if stack is empty else false
```

Nearest Smaller Element on Left

Another problem that can be solved using a stack is to find the nearest smaller element on the left of each element in a given array. For example, consider the array [3, 1, 4, 2, 5, 1, 3]. The nearest smaller element on the left of the first element (3) is none, the nearest smaller element on the left of the second element (1) is none, the nearest smaller element on the left of the third element (4) is 1, and so on. The algorithm to find the nearest smaller element on the left of each element in a given array using a stack is as follows:

- Create an empty stack and an empty result array.
- Traverse the array from left to right.
- While the stack is not empty and the top element is greater than or equal to the current element, pop the top element from the stack.
- If the stack is empty, the nearest smaller element on the left of the current element is none. Otherwise, the nearest smaller element on the left of the current element is the top element of the stack.
- Push the current element to the stack.
- Append the result to the result array.

Code

```
nearest_smaller_element_on_left(array):
    stack = empty stack
    result = empty array
    for each element e in array:
        while stack is not empty and peek(stack) >= e:
            pop(stack)
        if stack is empty:
            result.append(None)
        else:
            result.append(peek(stack))
        push(stack, e)
    return result
```