Principles of Computer Science II

Abstract Data Types

Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 16

Queues

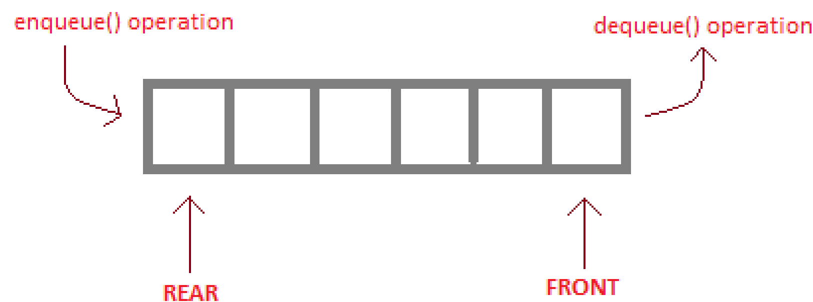
Queue is also an abstract data type or a linear data structure, in which

- the first element is inserted from one end called REAR (also called tail),
- and the deletion of existing element takes place from the other end called as FRONT(also called head).

This makes queue as FIFO(First in First Out) data structure, which means that element inserted first will also be removed first.

- The process to add an element into queue is called Enqueue.
- the process of removal of an element from queue is called Dequeue.

Queues



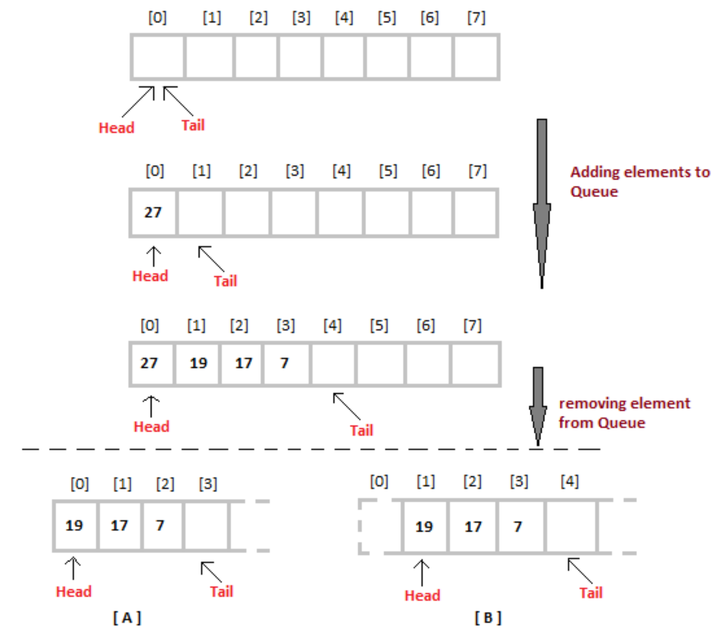
Basic features of Queues

1. Like Stack, Queue is also an ordered list of elements of similar data types.
2. Queue is a FIFO (First in First Out) structure.
3. Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element.
4. peek() function is oftenly used to return the value of first element without dequeuing it.

Implementation of Queues

- Queue can be implemented using an Array, Stack or Linked List.
- The easiest way of implementing a queue is by using an Array.
- Initially the head(FRONT) and the tail(REAR) of the queue points at the first index of the array (starting the index of array from 0).
- As we add elements to the queue, the tail keeps on moving ahead, always pointing to the position where the next element will be inserted, while the head remains at the first index.

Queues: An Example



Queues Code: Initialization

```
1 class Error(Exception):
2     """Base class for exceptions in this module."""
3     pass
4
5 class QueueError(Error):
6     def __init__(self, expression, message):
7         self.expression = expression
8         self.message = message
9
10 class Queue(object):
11     def __init__(self):
12         self.content = []
13
14     def size(self):
15         return len(self.content)
16
17     def isEmpty(self):
18         return not bool(self.content)
```

Queues: Algorithm for ENQUEUE operation

1. Check if the queue is full or not.
2. If the queue is full, then raise overflow error and exit the program.
3. If the queue is not full, then increment the tail and add the element.

```
1     def enqueue(self, value):
2         return self.content.append(value)
```

Queues: Algorithm for DEQUEUE operation

1. Check if the queue is empty or not.
2. If the queue is empty, then raise underflow error and exit the program.
3. If the queue is not empty, then return the element at the head and increment the head.

```
1     def dequeue(self):
2         if self.content:
3             return self.content.pop()
4         else:
5             raise QueueError("Queue is Empty")
```

Queues: Help function

```
1     def __repr__(self):
2         if self.content:
3             return '{}'.format(self.content)
4         else:
5             return "Queue empty!"
```

Queues: Testing

```
1 if __name__ == '__main__':
2     queue = Queue()
3     print("Is the queue empty? ", queue.isEmpty())
4     print("Adding 0 to 10 in the queue...")
5     for i in range(10):
6         queue.enqueue(i)
7     print("Queue size: ", queue.size())
8     print("Queue peek : ", queue.peek())
9     print("Dequeue...", queue.dequeue())
10    print("Queue peek: ", queue.peek())
11    print("Is the queue empty? ", queue.isEmpty())
12
13    print("Printing the queue...")
14    print(queue)
```

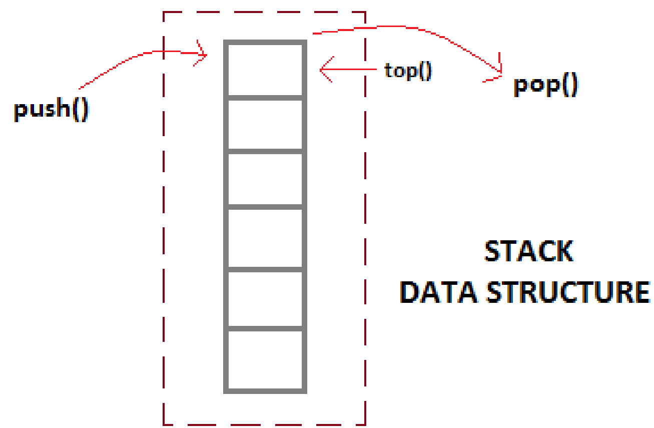
Stacks

Stack is an abstract data type with a bounded(predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order.

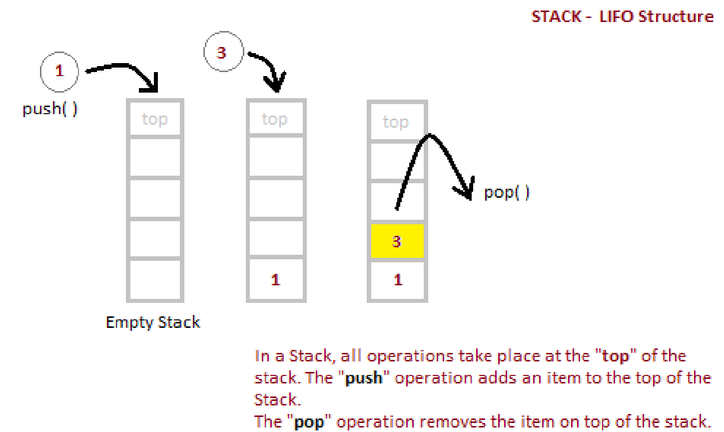
- Every time an element is added, it goes on the top of the stack,
- the only element that can be removed is the element that was at the top of the stack, just like a pile of objects.

- The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.
- Parsing, Expression Conversion(Infix to Postfix, Postfix to Prefix etc) and many more.

Stacks



Stacks: An Example



Basic features of Stacks

1. Stack is an ordered list of similar data type.
2. Stack is a LIFO structure. (Last in First out).
3. push() function is used to insert new elements into the Stack and
4. pop() function is used to delete an element from the stack.
5. Both insertion and deletion are allowed at only one end of Stack called Top.
6. Stack is said to be in Overflow state when it is completely full and is said to be in Underflow state if it is completely empty.

Stacks Code: Initialization

```
1 class Error(Exception):
2     pass
3
4 class StackError(Error):
5     def __init__(self, expression, message):
6         self.expression = expression
7         self.message = message
8
9 class Stack(object):
10    def __init__(self, size):
11        self.content = []
12        self.size = size
13
14    def size(self):
15        return len(self.content)
16
17    def isEmpty(self):
18        return not bool(self.content)
```

Stacks: Algorithm for PUSH operation

1. Check if the stack is full or not.
2. If the stack is full,then print error of overflow and exit the program.
3. If - the stack is not full, then increment the top and add the element.

```
1     def push(self, value):
2         if len(self.content) >= self.size:
3             raise StackError(self, "Overflow")
4 
5         self.content.append(value)
```

Stacks: Algorithm for POP operation

1. Check if the stack is empty or not.
2. If the stack is empty, then print error of underflow and exit the program.
3. If the stack is not empty, then print the element at the top and decrement the top.

```
1     def pop(self):
2         if self.content:
3             value = self.content.pop()
4             return value
5         else:
6             raise StackError(self, "Empty List")
```

Stacks: Testing

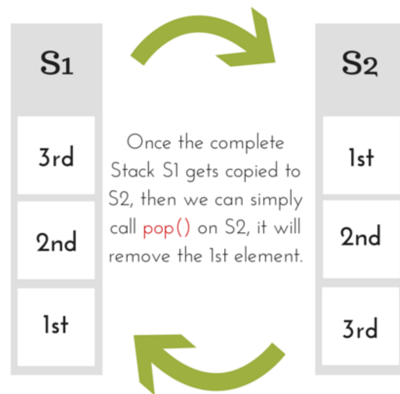
```
1 if __name__ == '__main__':
2     q = Stack(5)
3 
4     for i in range(15,20):
5         q.push(i)
6     for i in range(10,5,-1):
7         q.push(i)
8 
9     for i in range(1, 13):
10        print(q.pop())
```

Stacks: Testing with Error handling

```
1 if __name__ == '__main__':
2     q = Stack(5)
3 
4     try:
5         for i in range(15,20):
6             q.push(i)
7         for i in range(10,5,-1):
8             q.push(i)
9     except StackError:
10        print("Stack is full")
11 
12    try:
13        for i in range(1, 13):
14            print(q.pop())
15    except StackError:
16        print("Stack is empty")
```

Queues implemented with Stacks

Pop elements from S1 and push into S2,
int x = S1.pop();
S2.push(x);



Then push back elements to S1 from S2.