Alexandria University

Faculty of Engineering

Communication and Electronics Department

DS Sheet 4

Queues and Trees

Name/ Mostafa Ahmed Mohamed Rashed
ID/ 19017528
Section/ 3

1. Suppose an initially empty queue Q has performed a total of 32 enqueue operations, 10 front operations and 15 dequeue operations, 5 of which generated `EmptyQueueExceptions`, which were caught and ignored. What is the current size of Q?

Size = 32 – (15-5) = 22

2. Consider the array-based implementation of the queue ADT.

1. Suppose that you used the index variable r to point to the index after the last element in the queue. The front of the queue is the element at the index 0.Write algorithms to enqueue and dequeue elements from the queue. What is the problem with this implementation?
2. Alternatively, you would store the index of the first and last elements of the queue by the variables f and r. What are the values of f and r when: i. The queue is empty. ii. The queue is full. What is the problem with this scheme?
3. You then chose to solve the problem using a circular array (you would store the index of the first and last elements of the queue by the variables f and r and let f and r wrap around the array). What are the values of f and r when: i. The queue is empty. ii. The queue is full. What is the problem with this scheme?
4. You then chose to solve the problem by not allowing the last slot to be filled. How would you know that the queue is full and empty? Write the pseudocode for the Queue operations (use mod operations).

1.

Algorithm enqueue(o):
If r == q.size()
  Throw Exception
q[r] ← n
r ← r+1
	Algorithm dequeue(o):
	If r == 0
	  Throw Exception
	Temp ← q[0]
	for I ← 0 :r-1
	  q[i] = q[i+1]
	r ← r-1
	retutn temp

- Complexity = O(n)	

2.

Q is empty → f = r	Q is full → r = N
- Fixed size so we cannot add elements although the rest of array maybe empty - May be a waste of memory as the number of empty slots N-(r-f) may be at the front of the queue so they will never be filled	

3.

Q is empty → f = r	Q is full → f = r
- We cannot differentiate between empty and full	

4.

Q is empty → f = r	Q is full → f = r+1 or r = N-1 & f = 0

- Algorithm size():
  return (N-f+r)%N
- Algorithm isEmpty():
  return f == r
- Algorithm front():
  If isEmpty()
  Throw Exception
  return q[f]
- Algorithm enqueue():
  If (f == r+1) or (f == 0 and r = N-1)
  Throw Exception
  q[r] ← N
  r ← (r+1)%N
- Algorithm dequeue():
  If isEmpty():
  Throw Exception
  temp ← q[f]
  q[f] ← null
  f ← (f+1)%N
  return temp

3. Suppose you have a stack S containing n elements and a queue Q that is initially empty. Describe how you can use Q to scan S to see if it contains a certain element x, with the additional constraint that your algorithm must return the elements back to S in their original order. You may not use an array or linked list. Use only S and Q and a constant number of reference variables.

```
Algorithm Find(Q, S, x):
Flag ← false
for i ← 0:n-1
   temp = S.pop()
   if temp == x
      flag = true
   Q.enqueue(temp)
for i ← 0:n-1
   S.push(Q.dequeue())
for i ← 0:n-1
   Q.enqueue(S.pop())
for i ← 0:n-1
   S.push(Q.dequeue())
return flag
```

4. Show how to implement a stack using two queues.

- Algorithm push(o)
  q1.enqueue(o)

- Algorith pop()
  If q1.isEmpty() || q2.isEmpty()
    Throw Exception
  temp = q1.dequeue()
  q2.enqueue(temp)
  while ! (q1.isEmpty())
    q2.enqueue(q1.dequeue())
  temp = q2.dequeue()
  q1.enqueue(temp)
  while ! (q2.isEmpty())
    q2.enqueue(q1.dequeue())
  return temp

5. Given a priority queue with the interface:

```
void Insert(Item i, int priority);
Item DeleteMax();
```

Describe a how to implement a stack with the interface:

```
void Push(Item i)
Item Pop()
```

Static int priority = 0

Void Push(Item i){

Q.Insert(I, priority);

Priority++}

Item Pop(){

Priority--; return Q.DeleteMax()}

6. You are given an array A of size N. You can perform an operation in which you will remove the largest and the smallest element from the array and add their difference back into the array. So, the size of the array will decrease by 1 after each operation. You are given Q tasks and in each task, you are given an integer K. For each task, you have to tell the sum of all the elements in the array after K operations.

Sample input:

```
5 2 ← Array size is 5 and number of queries is 2
3 2 1 5 4 ← The array elements
1 ← First query
2 ← Second query
```

Sample output:

```
13 ← Subtract 5 - 1 = 4, the result array will be 3, 2, 4, 4 with sum 13
9 ← Subtract 5 - 1 = 4, 4 - 2 = 2, the result array will be 3, 2, 4 with sum 9
```

What data structure would you use to solve this problem? Think of one, or invent one, then write a pseudo code for this problem.

- Use Queues
-

```
// create a queue from the array
for x in A
   queue.enqueue(x)

// for each query
for i = 0:q - 1
 // read in the value of k
 K = input()
 // create a temp of the queue
Tqueue = queue

 // perform k operations on the copy
 for j = 0 :k - 1
  if tqueue.size()<= 1
    break
 //find the minimum and max element in temp and remove them
```

```
Min = inf
Max = 0
for x in tqueue:
   if x < min
      min = x
   if x > max
       max = x
tqueue.remove(min)
tqueue.remove(max)
tqueue.enqueue(max-min)
 // sum the elements in the temp
 set sum to 0
 for x in temp
  add x to sum
 print sum
```

7. An airport is developing a computer simulation of air-traffic control that handles events such as landings and takeoffs. Each event has a time-stamp that denotes the time when the event occurs. The simulation program needs to efficiently perform the following two fundamental operations:

1. Insert an event with a given time-stamp (that is, add a future event)
2. Extract the event with smallest time-stamp (that is, determine the next event to process).

Which data structure should be used for the above operations? ***Why****?*

priority queue, as it automatically sorts the events as we provide the time stamp as a key so we will always find the event with the smallest time stamp at the front of queue

8. Write an algorithm that counts the number of leaf nodes in a binary tree. And another algorithm to count the number of internal nodes in the binary tree.

-

```
Algorithm countLeafNodes(Node root):
if (root == null) return 0;
if (root.left == null && root.right == null)   return 1;
int count = 0;
count += countLeafNodes(root.left);
count += countLeafNodes(root.right);
return count;
```

-

```
Algorithm countInternalNodes(Node root):
if (root == null) return 0;
if (root.left == null && root.right == null)  return 0;
int count = 1;
count += countInternalNodes(root.left);
count += countInternalNodes(root.right);
return count;
```

9. Two binary trees are equivalent if they both have the same structure with same data in corresponding nodes. Write an algorithm to test whether two binary trees T1 and T2 are Equivalent.

```
Algorithm areTreesEqual(Node t1, Node t2):
if (t1 == null && t2 == null) return true;
if (t1 == null || t2 == null) return false;
if (t1.val != t2.val)return false;
return areTreesEqual (t1.left, t2.left) && areTreesEqual (t1.right, t2.right);
```

10. Write an algorithm SwapTree(T) that takes a binary tree and swaps the left and right children of every node.

```
Algorithm swapTree(Node T):
if (T == null) return;
   temp = T.left;
   T.left = T.right;
   T.right = temp;
   swapTree(T.left);
   swapTree(T.right);
```