Variants of ADT, Queue and their Novel Algorithms, Using ADT, List

O.E. Oguike
Department of Computer Science
University of Nigeria
Nsukka, Nigeria
osondu.oguike@unn.edu.ng

1Corresponding Author

ABSTRACT

Abstract— Queue as a data structure has important applications to computer science, which includes, different types of long term and short term operating system scheduling algorithms can be implemented using queue, printers connected to a computer network receive print jobs in a queue. As an abstract data type, the specifications of the operations of queue are separated from the implementation. Queue is a linear structure, which can be defined and implemented using different linear structures, like list, array etc. Using the abstract data type, list to implement a queue implies using the operations of the abstract data type, list to design the algorithms for the operations of the abstract data type, queue. This paper uses the ADT, list to present two variants of the abstract data type, queue. For each type of queue, it uses the algorithms for the operations of list to design the algorithms for the operations of the abstract data type, queue. It also designs the algorithms for the operations of queue of pairs by using the algorithms for the operations of list of pairs.

Keywords: Queue data structure, queue operations, list data structure, list operations, list of pairs, queue of pairs, abstract data type.

1. INTRODUCTION

Queue as an abstract data type can be used to store, access and manipulate a collection of data that are of a particular type. Example, a queue can be used to store, access and manipulate a collection of characters, integers or records of data. Using a list to represent a queue and using the algorithms for the operations of abstract data type list to define the algorithms for the operations of abstract data type, queue, two types of queue can be distinguished, which are, queue whose items are added from the left or queue whose front is on the right hand side, and queue, whose items are added from the right or queue whose front is on the left hand side. Figure 1 and figure 2 illustrate the two types of queue under consideration.

Figure 1: Queue whose items are added from the left or queue, whose front is on the right hand side.

Figure 2: Queue whose items are added from the right or queue, whose front is on the left hand side.

Illustrating further, these two types of queues can be used to represent a two-way road traffic system as shown in figure 3.

Figure 3: Illustration, using two-way road traffic system
2. SURVEY OF RELATED LITERATURE

Abstract data types have been extensively discussed in literature. It has been defined in [1] as an abstract model of data objects with a formal encapsulation of a logical architecture and valid operations of the data object. They identified a number of ADTs, which include, stack, queue, sequence, record, array, list, tree, file and graph. For the ADT, queue, they identified and specified the following static and dynamic operations of the queue abstract data type, create, enqueue, serve, clear, emptTest, fullTest, release. The operations of the abstract data type queue were identified and specified in [2] as empty, add, isempty, front, back. In [2], the operations of the ADT list were specified and identified as cons, head, tail, empty and isempty. The author in [2] identified two ways of designing the algorithms for the operations of a queue, which are, static and dynamic approaches. The static approach, according to her involves the use of array, while the dynamic approach involves the use of linked or chained implementation.

The chained or linked implementation, which the author focused on, used two pointers to define the queue structure. It did not use the operations of a linked list, like head, cons, tail, empty and isempty to defined and implement the algorithms for the operations of a queue, neither did she identify two types of queues, which are the focus of this paper. Data abstraction was considered in [4] as “the principle of specifying a data type together with its characteristic operations, in such a way that the internal structure of the data is kept hidden and that the specification has a clear meaning independent of the context in which it is used.” They pointed out some of the advantages of data abstraction, which includes, useful paradigm for the construction of stable software modules [8], layered software architecture [9], [10]. On the implementation of abstract data types, [3] remarked that object oriented programming achieves data abstraction by the use of procedural abstraction, while abstract data types depend on type abstractions. Furthermore, an abstract data type called search queues was considered as a blend of finite maps (dictionaries) and priority queues, the authors presented a simple implementation of the abstract data type and gave two applications of the abstract data type that demonstrate its usefulness. In [6], an approach for specifying the behaviour of software components that implements data abstraction was presented. According to them, “In a data abstraction, the internal state is hidden; clients can interact with the component by invoking the operations exposed by its interface.” Though the ADT, queue has been used in [7] for reservation of books in an academic library, in this paper, it is used to demonstrate how numeric data can be stored, assessed and processed in a queue.

3. DESIGN OF ALGORITHMS FOR THE OPERATIONS OF ABSTRACT DATA TYPE LIST

Since the underlying abstract data type for the two types of queues under consideration is list, this means that the algorithms for the operations of the abstract data type, list will be used to design the algorithms for the operations of the two types of queues. Therefore, it is necessary that the algorithms for the operations of the abstract data type, list be designed. The list structure will be defined as a list of real numbers, in a java class called listreal as follows:

```java
class listreal {
    private double data;
    private listreal link;
}
```

The algorithms for the operations of the abstract data type, list can be defined as follows:

- listreal emptylist() {
  1. Determine emptylist
  1.1 emptylist = null;
  2. Display emptylist
}

- listreal cons(double x, listreal l) {
  1. Request x
  2. Request l
  2.1 Declare and allocate, temp of the type listreal
  2.1.1 temp.data = x;
  2.1.2 temp.link = l;
  2.1.3 return temp;
  3. Display cons
}

- boolean isempty(listreal l) {
  1. Request l
  2. Determine isempty
  2.1 IF l = null THEN
  2.1.1 isempty = true;
  ELSE
  2.1.2 isempty = false
  3. Display isempty
}

- listreal head(listreal l) {
  1. Request l
  2. Determine head
  2.1 IF isempty(l) THEN
  2.1.1 head = “Empty list does not have any data”;
  ELSE
  2.1.2 head = l.data
  3. Display cons;
}
```

The algorithm, cons, makes a new node of the defined liststructure, and put x in the data part of temp and l in the link part of temp, and returns temp as the new list.
3. Display head
The algorithm, head takes a list of real numbers as parameter and it returns the real number that is on the left hand side of the list.

Listreal tail(listreal l)
1 Request l
2 Determine tail
2.1 IF isempty(l) THEN
2.1.1 tail = l;
ELSE
2.1.2 tail = l.link
3 Display tail

The algorithm, tail takes a list of real numbers as parameter and it returns the list, l, if it is empty, otherwise, it removes the head and returns the list that remain.

4. DESIGN OF ALGORITHMS FOR THE OPERATIONS OF A QUEUE WHOSE FRONT IS ON THE RIGHT HAND SIDE

The algorithms for the operations of a list, which have been defined earlier will be used to define algorithms for the operations of the abstract data type, queue, whose front is on the right hand side, i.e. items are added into the queue from the left hand side. The algorithms for the operations of this type of queue are: emptyqueue, isemptyqueue, frontqueue, backqueue and addqueue. The following are the design of these algorithms.

Queue real1 queuempty()
1 Determine queuempty
1.1 Queuempty = emptylist()
2 Display queuempty

The algorithm queuempty takes nothing as parameter, and it uses the algorithm for the operation of a list, called emptylist to return an empty queue.

Boolean isemptyqueue(queue real1 q)
1. Request q
2 Determine isemptyqueue
2.1 isemptyqueue = isempty(q)
3 Display isemptyqueue

The algorithm isemptyqueue takes a queue of real numbers as parameter and it uses the algorithm for the operation of a list of real numbers, called isempty to return true if the queue is empty, otherwise, it returns false.

double frontqueue(queue real1 q)
1. Request q
2. Determine frontqueue
2.1 IF isemptyqueue(q) THEN
2.1.1 frontqueue = "Empty queue does not have front."
ELSE
2.1.2 IF isemptyqueue(tail(q)) THEN
2.1.2.1 frontqueue = head(q)
ELSE
3 Display frontqueue

The algorithm frontqueue takes a queue of real numbers as parameter, and it uses the algorithm for the operation of list and queue to return the item that is in the front of the queue, if it exists, otherwise, it returns an appropriate error message.

Queue real1 backqueue(queue real1 q)
1. Request q
2. Determine backqueue
2.2 IF isemptyqueue(q) THEN
2.2.1 backqueue = q
ELSE
2.2.2 IF isemptyqueue(tail(q)) THEN
2.2.2.1 backqueue = emptyqueue()
ELSE
2.2.2.2 backqueue = cons(head(q), backqueue(tail(q)))
3 Display backqueue

The algorithm backqueue takes a queue of real numbers as parameter and it uses the algorithms for the operations of list and queue to determine the back of the queue.

Queue real1 addqueue(double x, queue real1 q)
1 Request data
1.1 Request x
1.2 Request q
2 Determine addqueue
2.1 addqueue = cons(x, q)
3 Display addqueue

The algorithm addqueue uses the algorithm for the operation of a list to add an item into the queue from the left hand side.

5. DESIGN OF ALGORITHMS FOR THE OPERATIONS OF A QUEUE WHOSE FRONT IS ON THE LEFT HAND SIDE

This section aims at considering another way to look at a queue, i.e. a queue whose front is on the left hand side, items are added into the queue from the right. Using the algorithms for the operations of list to design the algorithms for the operations of a queue, whose front is on the left will produce different algorithms from the algorithms for the operations of queue whose front is on the right. The following are the design of algorithms for the operations of this type of queue.

double frontqueue(queue real2 q)
1. Request q
2. Determine frontqueue
2.1 IF isemptyqueue(q) THEN
2.1.1 frontqueue = “Empty queue does not have front.”
ELSE
2.1.2 IF isemptyqueue(tail(q)) THEN
2.1.2.1 frontqueue = head(q)
ELSE

The algorithm frontqueue takes a queue of real numbers as parameter and it uses the algorithm for the operation of list and queue to return the item that is in the front of the queue, if it exists, otherwise, it returns an appropriate error message.
2.1.2 \[\text{frontqueue} = \text{head}(q)\]

3. Display frontqueue
This algorithm can be compared with the algorithm called frontqueue in section 4, in order to identify the difference.

Queue\text{real2} backqueue(\text{queuereal2} q)
1. Request \text{q}
2. Determine backqueue
\hspace{0.5cm} 2.1 \text{IF isemptyqueuet}(q) \text{THEN}
\hspace{1cm} 2.1.1 \text{backqueue} = q
\hspace{1cm} ELSE
\hspace{1cm} 2.1.2 \text{backqueue} = \text{tail}(q)

3. Display backqueue
This algorithm can be compared with the algorithm, called backqueue in section 4, in order to identify the difference.

Queue\text{real2} addqueue(double x, \text{queuereal2} q)
1. Request data
2. Determine addqueue
\hspace{0.5cm} 2.1 \text{IF isemptyqueue}(q) \text{THEN}
\hspace{1cm} 2.1.1 \text{addqueue} = \text{cons}(x, q)
\hspace{1cm} ELSE
\hspace{1cm} 2.1.2 \text{addqueue} = \text{cons}((\text{front}(q),
\hspace{1cm} \text{addqueue}(x,\text{backqueue}(q)))

3. Display addqueue
This algorithm, addqueue can be compared with the algorithm called addqueue in section 4, in order to identify the difference. The two algorithms, emptyqueue and isemptyqueue will be the same as the emptyqueue and isemptyqueue algorithms in section 4.

6. DESIGN OF ALGORITHMS FOR THE OPERATIONS OF ABSTRACT DATA TYPE LIST OF PAIR

Having designed the fundamental algorithms for the operations of list of real numbers, which have been used to design the fundamental algorithms for the operations of queue of single real numbers, this can be extended to queue of pair of real numbers. Before the algorithms for the operation of queue of pair of real numbers will be designed, the algorithms for the operations of list of pair of real numbers will be developed, this is because queue of pair of real numbers will be considered as a list of pair of real numbers. The structure of list of pair of real numbers will be defined in a java class, listpair as:

- Double datax
- Double datay
- Listpair link

The following are the operations of the list of pair of real numbers, emptypair, isemptypair, conspair, headx, heady and tailpair. The algorithms for the operations of list of pair of real numbers can be defined as follows:

listpair emptypair()
1. Determine emptypair
\hspace{0.5cm} 1.2 \text{emptypair} = \text{null};

2. Display emptypair

boolean isemptypair(listpair l)
1. Request \text{l}
2. Determine isemptypair
\hspace{0.5cm} 2.1 \text{IF l = null THEN}
\hspace{1cm} 2.1.1 \text{isemptypair} = \text{true};
\hspace{1cm} ELSE
\hspace{1cm} 2.1.2 \text{isemptypair} = \text{false}

3. Display isemptypair

listpair conspair(double x, y, listpair l)
1. Request data
2. Determine conspair
\hspace{0.5cm} 2.1 \text{Declare and allocate, temp of the type listpair}
\hspace{1cm} 2.2 temp.datax = x;
\hspace{1cm} 2.3 temp.datay = y;
\hspace{1cm} 2.4 temp.link = l;
\hspace{1cm} 2.5 conspair = temp

3. Display conspair;

double headx(listpair l)
1. Request l
2. Determine headx
\hspace{0.5cm} 2.1 \text{IF isemptypair(l) THEN}
\hspace{1cm} 2.1.1 headx = “Empty list does not have any data”;\n\hspace{1cm} ELSE
\hspace{1cm} 2.1.2 headx = l.datax

3. Display headx

double heady(listpair l)
1. Request l
2. Determine heady
\hspace{0.5cm} 2.1 \text{IF isemptypair(l) THEN}
\hspace{1cm} 2.1.1 heady = “Empty list does not have any data”;\n\hspace{1cm} ELSE
\hspace{1cm} 2.1.2 heady = l.datay

3. Display heady

Listpair tailpair(listpair l)
1. Request l
2. Determine tailpair
\hspace{0.5cm} 2.1 \text{IF isemptypair(l) THEN}
\hspace{1cm} 2.1.1 tailpair = \text{l};
\hspace{1cm} ELSE
\hspace{1cm} 2.1.2 tailpair = \text{l.link}

3. Display tailpair

158
7. DESIGN OF ALGORITHMS FOR THE OPERATIONS OF ABSTRACT DATA TYPE QUEUE OF PAIR, WHICH HAS ITS FRONT ON THE RIGHT HAND SIDE

As an extension of figure 1, a queue of pairs, which has the fronts on the right hand side, can be illustrated using the diagram in figure 4. In this type of queue, items are added into the queue in pairs.

As an extension of figure 1, a queue of pairs, which has the fronts on the right hand side, can be illustrated using the diagram in figure 4. In this type of queue, items are added into the queue in pairs.

Figure 4: Queue of pairs, which has fronts on the right hand side.

The various algorithms for the operations of list of pairs of real numbers will be used to design these algorithms.

```
Queuepair1 queuemptypair()
1. Determine queuemptypair
   1.1 Queuempty = emptypair()
2. Display queuemptypair

Boolean isemptypqueuepair(queuepair1 q)
1. Request q
2. Determine isemptypqueuepair
   2.1 isemptypqueuepair = isemptypair(q)
3 Display isemptypqueuepair

double frontqueuex(queuepair1 q)
1. Request q
2. Determine frontqueuex
   2.1 IF isemptypqueuepair(q) THEN
      2.1.1 frontqueuex = "Empty queue does not have front."
   ELSE
      2.1.2 IF isemptypqueuepair(tailpair(r(q))) THEN
         2.1.2.1 frontqueuex = headx(q)
      ELSE
         2.1.2.2 frontqueuex = frontqueuex(tailpair(q))
3 Display frontqueuex

double frontqueuey(queuepair1 q)
1. Request q
2. Determine frontqueuey
   2.1 IF isemptypqueuepair(q) THEN
      2.1.1 frontqueuey = "Empty queue does not have front."
   ELSE
      2.1.2 IF isemptypqueuepair(tailpair(r(q))) THEN
         2.1.2.1 frontqueuey = heady(q)
      ELSE
         2.1.2.2 frontqueuey = frontqueuey(tailpair(q))
3 Display frontqueuey

Queuepair1 backqueuepair(queuepair1 q)
1. Request q
2. Determine backqueuepair
   2.1 IF isemptypqueuepair(q) THEN
      2.1.1 backqueuepair = q
   ELSE
      2.1.2 IF isemptypqueuepair(tailpair(r(q))) THEN
         2.1.2.1 backqueuepair = emptypqueuepair()
      ELSE
         2.1.2.2 backqueuepair = conspair(headx(q), heady(q), backqueuepair(tailpair(q)))
3 Display backqueuepair

Queuepair1 addqueuepair(double x, y, queuepair1 q)
1 Request data
   1.1 Request x
   1.2 Request y
   1.3 Request q
2 Determine addqueuepair
   2.1 addqueuepair = conspair(x, y, q)
3 Display addqueuepair
```

```
8. DESIGN OF ALGORITHMS FOR THE OPERATIONS OF ABSTRACT DATA TYPE, QUEUE OF PAIR, WHICH HAS ITS FRONT ON THE LEFT HAND SIDE

As an illustration, a queue of pairs, which has the fronts on the left hand side, can be illustrated using the diagram in figure 5. Items are added into the queue in pairs.

Front

Figure 5: Queue of pairs, which has fronts on the left hand side.

The various algorithms for the operations of list of pairs of real numbers will be used to design the following algorithms for the queue of pairs of real numbers, which has fronts on the left hand side.

double frontqueuex(queuepair2 q)
1. Request q
2. Determine frontqueuex
2.1 IF isemptyqueuepair(q) THEN
2.1.1 frontqueuex = “Empty queue does not have front”
ELSE
2.1.2 frontqueuex = headx(q)
3. Display frontqueuex

double frontqueuey(queuepair2 q)
1. Request q
2. Determine frontqueuey
2.1 IF isemptyqueuepair(q) THEN
2.1.1 frontqueuey = “Empty queue does not have front”
ELSE
2.1.2 frontqueuey = heady(q)
3. Display frontqueuey

queuepair2 backqueuepair(queuepair2 q)
1. Request q
2. Determine backqueuepair
2.1 IF isemptyqueuepair(q) THEN
2.1.1 backqueuepair = q
ELSE
2.1.2 backqueuepair = tailpair(q)
3. Display backqueuepair

Queuepair2 addqueuepair(double x, y, queuepair2 q)
1. Request data
1.1 Request x
1.2 Request y
1.3 Request q
2. Determine addqueuepair
2.1 IF isemptyqueuepair(q) THEN
2.1.1 addqueuepair = conspair(x, y, q)
ELSE
2.1.2 addqueuepair = conspair(frontx(q), fronty(q), addqueuepair(x, y, backqueuepair(q)))
3. Display addqueuepair

9. APPLYING THE ALGORITHMS FOR OPERATIONS OF QUEUE OF PAIRS OF REAL NUMBERS TO STATISTICAL COMPUTATION

The algorithms for the operations of queue of pairs of real numbers can be used to define algorithms that will perform statistical computation. Using any of the types of queue, i.e. queue, whose front is on the right hand side or queue, whose front is on the left hand side, the same algorithms that perform the statistical computation will produce the same result. Suppose pairs of real number in form of \((X_i, Y_i)\) are added into the queue of pairs with front on the right hand side or queue of pairs with front on the left hand side. The task is to use the algorithms for the operations of queue of pair of real number to compute the beta statistics, which is defined using this statistical formula.

\[
\left( \frac{\sum_{i=1}^{n} X_i Y_i - \left( \sum_{i=1}^{n} X_i \right) \left( \sum_{i=1}^{n} Y_i \right)}{\left( \sum_{i=1}^{n} X_i^2 - \left( \sum_{i=1}^{n} X_i \right)^2 \right) / (n-1)} \right)
\]

In order to define the algorithm that will compute the statistic, the following algorithms will be used:

Double sumpro(queuepair1 q)
1. Request q
2. Determine sumpro
2.1 IF isemptyqueuepair(q) THEN
2.1.1 sumpro = 0
ELSE
2.1.2 sumpro = fronth(queuepair(q) + sumpro(backqueuepair(q)))
3. Display sumpro

double sumx(queuepair1 q)
These algorithms can be used to define the algorithm that will compute the above statistic, as follows:

Double beta(queuepair1 q)
1 Request q
2 Determine beta
2.1 top = count(q)*sumpro(q) – sumx(q)*sumy(q)
2.2 bottom = count(q)*sumsqx(q) – sumx(q)*sumx(q)
2.3 beta = top/bottom
3 Display beta

10. IMPLEMENTATION OF THE NOVEL ALGORITHMS

Java programming language was used to implement all the novel algorithms for the operations of each type of queue considered in this paper. All the structured programming facilities that Java provides were used. Example, each of the algorithms was implemented as a Java method/function; each abstract data type was implemented in a separate java class with appropriate interface that was used to separate the specification from the implementation. The algorithms for the operations of the fundamental abstract data type, list of single real number were implemented in a Java class, called listreal, while the algorithms for the operations of the fundamental abstract data type, queue of single real numbers, whose front is on the right were implemented in a java class, called queuereal1, and the algorithms for the operations of fundamental abstract data type, queue of single real numbers, whose front is on the left hand side were implemented in a Java class called queuereal2. The two classes, queuereal1 and queuereal2 interacted with the class, listreal using the concepts of inheritance and composition. A test program was written to test all the implemented algorithms for the operations of the two types of fundamental queues of real number. Figure 4 shows the class diagram that illustrates the implementation of two fundamental queues of single real numbers.
In a similar manner, the algorithms for the operations of the two types of extended queues, i.e. queue of pairs of real numbers, whose front is on the right and queue of pairs of real numbers, whose front is on the left, were implemented using the algorithms for the operations of the extended list of pairs of real number. The algorithms for the operations of list of pairs of real numbers were implemented in a java class called listpair, while the algorithms for the operations of queue of pairs of real numbers, whose front is on the right hand side were implemented in a java class, called queuepair1.

The algorithms for the operations of queue of pairs of real numbers, whose front is on the left hand side were implemented in a java class, called queuepair2. The statistical computations algorithms for the two types of queues were implemented in two java classes called, queuestat1 and queuestate2, respectively. A test program was written, which was used to read pairs of real numbers into the two types of queues, and for each type of queue, the test program displayed the result of the beta statistics. Figure 5 shows the class diagram for the implementations of the two types of queues of pairs of real numbers and the statistical computation, using the two types of queues.
11. RESULTS OF THE IMPLEMENTED ALGORITHMS

For the two types of fundamental queues of single real numbers, the test program was used to test the implemented algorithms and the same result was obtained. Sample test data was used to test the implemented algorithms. Table 1 shows a sample test data that was used to test the implemented algorithms for the two types of queues of pairs of real numbers, and the same result was obtained as the value of beta. The result was also confirmed using EXCEL spreadsheet, as shown in figure 6.

When the test program was executed, using the test data, the values of beta obtained for the two types of queues were the same as the result shown in the confirmation result, using EXCEL spreadsheet.

12. CONCLUSION

This paper has been able to design novel algorithms for two types of queues, i.e. queue whose front is on the right hand side and queue whose front is on the left hand side. It has also extended these fundamental queues of single real number to queues of pairs of real numbers, and novel algorithms have been designed for these two types of queues of pairs of real numbers. Furthermore, the paper has been able to use these two types of queues of real numbers to design algorithms for a particular statistic, and the results obtained for the two types of queues are the same.
**REFERENCES**


**Author’s Bio**

Oguike, Osondu Everestus is a Senior Lecturer in the Department of Computer Science, University of Nigeria, Nsukka, Enugu State, Nigeria. He has received many academic prizes and scholarships as a result of his outstanding academic performance. He is interested in modeling the performance of parallel computer system. He can be reached by phone on +2348035405100 and through E-mail osondu.oguike@unn.edu.ng