ON THE IMPLEMENTATION OF DATA STRUCTURES THROUGH THEORY OF LISTS

B. K. Tripathy[1] and S. S. Gantayat[2]

[1]School of Computing Sciences & Engineering
VIT University, Vellore – 632 014, Tamilnadu, INDIA
[1]tripathybk@vit.ac.in
[2]Department of Computer Science & Engineering
GMR Institute of Technology, Rajam – 532 127, Andhra Pradesh, INDIA
[2]sasankosekhar.g@gmrit.org

ABSTRACT

The concept of a set can be defined through its characteristic function and that of a bag is defined through the count function. Following this approach, Tripathy, Ghosh and Jena introduced the concept of position function to define lists. The new definition has much rigor than the earlier one used in computer science in general and functional programming ([2]) in particular. Several of the concepts in the form of operations, operators and properties have been established in a sequence of papers by Tripathy and his coauthors ([4, 7, 8]. Also, the concepts of fuzzy lists ([5]) and that of intuitionistic fuzzy lists ([6]) have been defined and studied by them. Recently an application to develop list theoretic relational databases and operations on them has been put forth by Tripathy and Gantayat ([9]). In the present article we provide another application of this approach in defining data structures like Stack, Queue and Array and establishing operations on them. One of the major advantages of this approach is the ease in extending all the concepts for basic lists to the context of fuzzy lists [5] and intuitionistic fuzzy lists [6]. In this paper, we illustrate how this can be done. The notions of fuzzy lists and intuitionistic fuzzy lists are new and least attended concepts and the works in this paper opens a new application area to develop fuzzy and intuitionistic fuzzy data structures.

KEYWORDS

Lists, Fuzzy lists, Intuitionistic fuzzy lists, Stack, Queue, Array

1. INTRODUCTION

It is well known that the notion of a set and its characteristic function are interchangeable concepts. Similarly, the notion of a bag and its count function ([10]) are interchangeable concepts. So, it is natural to think of some such characteristic for lists, one of the most widely used data structures in the field of Computer Science. This could be realised by Tripathy, Ghosh and Jena ([4]), where they introduced the notion of position function for lists. As a continuation of their effort, they defined many concepts associated with lists through this new definition and also established several theorems which reveal the properties of lists. This study was further carried out in [7] and [8]. Just to recall, both the number of times elements occur and the order of their occurrence are important in a list. In a bag only the numbers of times elements occur are important. In the definition of a set the order of occurrence of elements is un-important and the repetition of elements is considered as illegal.

The notion of fuzzy sets introduced by Zadeh ([11]) is an important model to capture impreciseness in data. The notions of fuzzy bags and fuzzy lists were not considered in literature before these were defined by Tripathy et al [5]. The concept of intuitionistic fuzzy sets introduced by Atanassov ([1]) is an extension of the notion of fuzzy sets and is a better model than fuzzy sets to model impreciseness in data. As it is natural, the concept of intuitionistic fuzzy lists was defined and their properties were studied by Tripathy et al ([6]).

DOI : 10.5121/ijitcs.2012.2404

Recently, as an application of the new definition of lists, a framework of relational data model and some operations on it have been defined by Tripathy et al ([9]).In this article we study another application of lists in realizing data structures. A list is defined as a linearly ordered collection of elements of similar data types with single or multiple occurrences. Since stack, queue and array are linear data structures; we establish here as how these data structures can be implemented using lists and operations on them. The operations on these data structures are, generally, insertion or deletion of an element, searching an element or finding its position and replacement of an element at a particular position by another element.

Also, we shall define these data structures in their fuzzy and intuitionistic fuzzy versions. Perhaps these concepts are used by us for the first time in the sequel. To be precise, in this article we have shown how the operators and operations of lists, fuzzy lists and intuitionistic fuzzy list can be used in defining data structures like stack, queue, array and operations on them. This creates a background basing upon which packages on lists and their applications can be developed dealing with data structures. The important factor is that once the basic operations are developed using functional programming the development of packages becomes easy. Also, the corresponding fuzzy and intuitionistic fuzzy data structures, which are yet to be explored to their full potential can be realized and used with minor changes in the definitions of crisp lists and operations on them have been indicated.

The organization of the paper from this point onwards is as follows. We introduce some of the definitions of concepts associated with the new definition of list, which has been developed in different papers as mentioned above in section 2.We state two theorems to be used in the sequel. In section 3 we introduce several data structures using lists and operations on them. Also, we illustrate the execution of these operations through suitable examples. In the next section we introduce the definition of fuzzy lists and define data structures and operations on these data structures basing on them. But we restrict ourselves to only a few of these to optimise space utilizations. However, the other related concepts can be defined with little effort. In section 5, we work on intuitionistic fuzzy lists in a similar manner. In section 6 we provide some concluding remarks. We end up the paper with a bibliography of referred works.

2. DEFINITIONS AND PROPERTIES

Definition 2.1: A **list** L drawn from a set X is represented by a position function P_L, defined as

$$P_L: X \to P(N),$$

where P(N) denotes power set of the set of non–negative integers N.

Definition 2.2: For any finite list L drawn from X, the **cardinality** of L is denoted by #L and is defined as

$$\#L = \sum_{x \in X} |P_L(X)|,$$

whenever the right hand side exists. In this case L is said to be finite, otherwise L is said to be infinite.

Definition 2.3: For any finite list L drawn from X, the **reverse** of L, denoted by rev(L) is a list on X, and is given by the position function,

$$P_{rev(L)}(x) = \{\#L - 1 - t : t \in P_L(x)\} \text{ for each } x \in X.$$

Definition 2.4: For any finite list L drawn from X, the **head** of L is an element denoted by **hd**(L), where hd(L) = x if $0 \in P_L(x)$.

Definition 2.5: Let L be a list. We define the **tail** of L denoted by tl(L) as

$$P_{tl(L)}(x) = \{t - 1 : t \in P_L(x), t \neq 0\}.$$

Definition 2.6: For any list L we define the functions **init** and **last** as:

init(L) = rev(tl(rev(L))) and last(L) = x; if $(\#L-1)\in P_L(x)$.

Definition 2.7: For any $x\in X$ and a list L drawn from X, we denote the list which is obtained by adding an element x at the beginning of the list L by **cons(x, L)** and define it by its membership function,

$$P_{cons(x,L)}(y)=\begin{cases}\{r-1: r\in P_L(y),\ if\ \ y\neq x;\\ \{0\}\bigcup\{r+1: r\in P_L(x), if\ \ y=x.\end{cases}$$

Definition 2.8: A list L is **empty** if $P_L(x) = \varphi$ for each $x \in X$ and denoted as L = [ ].

Definition 2.9: Let L_1 and L_2 be two finite lists drawn from X. Then we define the **concatenation** of L_1 and L_2 denoted by $L_1 \# L_2$ and is given by the position function,

$$P_{(L_1 \# L_2)}(x) = P_{L_1}(x) \cup \left\{\#\, L_1 + t : t \in P_{L_2}(x)\right\}\ \forall x \in X.$$

Definition 2.10: Let L be finite list drawn from X and [x] be a singleton list. Then the position function of the list **L – [x]** is defined as follows

Case-I: $P_L(x) \neq \phi$

$$P_{L-[x]}(y) = \begin{cases}\left\{r : r\in P_L(y),\ r < \min P_L(x)\right\}\cup\left\{r - 1 : r \in P_L(y),\ r > \min P_L(x)\right\}, & \text{if } y \neq x;\\ \left\{r - 1 : r \in P_L(x),\ r > \min P_L(x)\right\}, & \text{if } y = x.\end{cases}$$

Case-II: $P_L(x) = \phi$

$$P_{L-[x]}(y) = P_L(y)$$

Definition.2.11: Let L_1 and L_2 be two lists drawn from X, then the **zip** of L_1 and L_2 is given as

$$P_{zip(L_1, L_2)}(x, y) = P_{L_1}(x) \cap P_{L_2}(y)$$

Definition.2.12: For any two finite lists L and L′ drawn from X, the **difference** of the lists L – L′ is given by its position function, defined recursively by

$$P_{L-L'}(x) = \begin{cases}P_L(x), & if\ \ L' is\ an\ empty\ list;\\ P_{\{L-[hd(l')]\}-tl(L')}(x), & otherwise.\end{cases}$$

Definition 2.13: Let L be a finite list drawn from X. Then the **take** operator on L, for any given $n \in N$, is denoted by take(n, L) and it is a list whose position function is given by

$$P_{take(n,L)}(x)=\begin{cases}\phi, if\ \ n\leq \min P_L(x)\ or\ P_L(x)=\phi;\\ \{r: r<n\ and\ r\in P_L(x)\}, if\ \ n>P_L(x).\end{cases}$$

Definition 2.14: Let L be a finite list drawn from X. Then, for any $n \in N$ the **drop** operator on L is denoted by drop (n, L) and it is a list whose position function is given by

$$P_{drop(n,L)}(x)=\begin{cases}\phi, & if\ n>\max P_L(x);\\ \{r-n: r\geq n\ and\ r\in P_L(x)\}, & if\ \ n\leq \max P_L(x).\end{cases}$$

Definition 2.15: For any list L drawn from X and any natural number n, we define the element **index(L, n)** as $x \in X$, such that x = hd(take(n+1,L) – take(n, L)).

The following two properties of lists, which are Theorems established in ([7]) are to be used in the coming sections:

Theorem 2.1: If L_1 and L_2 are the lists drawn from X and f is a mapping from X into X, then for any n ∈ N,

a) $$take(n, L_1 \mathbin{\#} L_2) = \begin{cases} take(n, L_1), & if\ n \le \#L_1; \\ L_1 \mathbin{\#} take(n - \#L_1, L_2), & if\ n > \#L_1. \end{cases}$$

b) $$drop\left(n,\ L_1 \mathbin{\#} L_2\right) = \begin{cases} drop\left(n,\ L_1\right) \mathbin{\#} L_2\ , & \text{if } n \le \#L_1; \\ drop\left(n - L_1,\ L_2\right)\ , & \text{if } n > \#L_1. \end{cases}$$

Theorem 2.2: For any n ∈ N and for lists L_1 and L_2 drawn from X,

$$index\left(L_1 \mathbin{\#} L_2,\ n\right) = \begin{cases} index\left(L_1, n\right), & \text{if } n < \#L_1; \\ index\left(L_2,\ n\ -\ \#L_1\right), & \text{if } n \ge \#L_1. \end{cases}$$

3. DATA STRUCTURES THROUGH LISTS

A list is a linearly ordered collection of elements of similar data types with single or multiple occurrences. Since stack, queue and array are linear data structures; we shall establish how these data structures can be implemented using lists and operations on them can be realized through operations on lists established mentioned in section 2 and by other authors. The operations on these data structures are, generally, insertion or deletion of an element, searching an element, or finding its position and replacement of an element at a particular position by another element.

We shall establish the applications of lists and properties on them by showing the representation and operations on these data structures in the following.

3.1. IMPLEMENTATION OF A STACK AND ITS OPERATIONS

A *stack* is a linearly ordered set of elements and the operation on elements in it follows the LIFO (Last In First Out) principle. The insertion and the deletion of an element is done from one end of this linear structure with the other end being fixed. A stack has a fixed size, which is the maximum number of elements, it can store in it. The beginning of the structure, which is a location in computer memory, is called the 'BOTTOM' of the Stack. At any instant of time there is a variable called STACK pointer, which contains the position of the latest element inserted into the stack. This position is called the 'TOP' of the stack. The process of inserting a new element into the stack is called PUSH and the process of deletion of an element from a stack is called POP, which is done from the TOP of the stack.

Suppose the size of a stack S at any point of time is #S and M is its maximum size. In the beginning we take S = [ ].

3.1.1 Insertion or addition of an element in S:

$$PUSH(x, S) = \begin{cases} S \mathbin{\#} [x]\ , & \text{if } \#S < M; \\ \text{Overflow}, & \text{if } \#S = M. \end{cases}$$

3.1.2 Deletion or removal of an element from S:

$$POP(S) = \begin{cases} init(S)\ , & \text{if } \#S > 0; \\ [\ ]\ , & \text{if } \#S = 0. \end{cases}$$

3.1.4 To get the top of the elements from S:

$$TOP(S) = last(S), \quad \text{if } \#S \neq 0.$$

3.1.5 To find an element at the i^{th} position from the top of S:

$$PEEP(S, i) = \begin{cases} hd(drop\ (\#S - i, S)) \\ or \\ last(take(\#S - i+1,\ S)); \end{cases}$$

where $0 \leq i \leq \#S$.

3.1.6 To check whether the stack S is empty or not:

EMPTY(S) = True , if #S = 0.

We shall illustrate the above operations in the following example.

Example 3.1: Consider the following list of five names.

S = [“John”, “Sonu”, “Rama”, “Samir”, “Tom”].

Here #S = 5. Let the list size M = 10. Then we can insert at most 5 names in the list.

(a) Suppose we want to insert a name x = “Anu” in the stack.

PUSH(“Anu”, S) = S ⫴ [“Anu”]

= [“John”, “Sonu”, “Rama”, “Samir”, “Tom”, “Anu”].

(b) POP(S) = delete the last element of the list S

= init[“John”, “Sonu”, “Rama”, “Samir”, “Tom”]

= [“John”, “Sonu”, “Rama”, “Samir”]

(c) PEEP(S,2) = hd(drop(3,S))

= hd([“Samir”, “Tom”])

= [“Samir”].

3.2. IMPLEMENTATION OF A QUEUE AND ITS OPERATIONS

A *Queue* is a linearly ordered set of elements which works on the FIFO (First In First Out) principle, that is, insertion or addition of elements take place at the end of the queue and deletion or removal of elements takes place from the beginning of the queue.

Suppose that a queue Q with the maximum size M, where $M > 0$ and Q contains #Q number of elements. Suppose x is an element of X, used in the operations.

3.2.1 Insertion or addition of an element into Q:

$$Q\text{-}INSERT(x, Q) = \begin{cases} Q \Cup [x] , & \text{if } \#Q < M ; \\ Overflow, & \text{if } \#Q \geq M \textit{ or size of } Q \text{ is fixed.} \end{cases}$$

3.2.2 Deletion or removal of an element from Q:

$$Q\text{-}DELETE(Q) = \begin{cases} tl(Q) , & \text{if } \#Q > 0; \\ underflow, & \text{otherwise.} \end{cases}$$

3.2.3 To get an element from Q:

a) Front element of Q:

$$FRONT(Q) = \begin{cases} hd(Q) , & \text{if } \#Q \neq 0; \\ [\ ] , & \text{otherwise.} \end{cases}$$

b) Rear element of Q:

$$REAR(Q) = \begin{cases} last(Q) , & \text{if } \#Q \neq 0; \\ [\ ] , & \text{otherwise.} \end{cases}$$

3.2.4 To check the Q is empty or not:

EMPTY(Q) = True , if #Q = 0.

Example 3.2: Consider the same example as given in Example 2.1., where the given list is a queue. That is, Q = [“John”, “Sonu”, “Rama”, “Samir”, “Tom”].

Here #Q = 5. Suppose we want to insert a name x = “Anu” in a queue.

a) Q-INSERT(“Anu”, Q) = Q ⫲ [“Anu”]

= [“John”, “Sonu”, “Rama”, “Samir”, “Tom”, “Anu”].

b) Q-DELETE(Q) = tl(Q)

= tl([“John”, “Sonu”, “Rama”, “Samir”, “Tom”])

= [“Sonu”, “Rama”, “Samir”, “Tom”].

c) FRONT(Q) = hd([“John”, “Sonu”, “Rama”, “Samir”, “Tom”]) = “John”

d) REAR(Q) = last([“John”, “Sonu”, “Rama”, “Samir”, “Tom”]) = “Tom”

3.3. IMPLEMENTATION OF ARRAYS AND OPERATIONS ON THEM

An *array* is a finite linearly ordered set such that an element can be inserted at any position and deleted from any position of the array.

An array is a finite list. Suppose A is an array of size M. The number of elements in an array A is #A. Suppose x is an element used in the operation. All the operations on arrays are basically operations on finite lists.

3.3.1 Insertion of an element

Insertion of an element can take place at the following positions

(i) at the beginning of an array
(ii) at the end of an array
(iii) at any position inside an array

(i) At the beginning of an array:

B-INSERT(x, A) = cons(x, A), if #A < M.

(ii) At the end of an array:

The operation is same as in the case of a Stack and a Queue.

E-INSERT(x, A) = A ⫲ [x], if #A < M.

(iii) At any position of an array:

INSERT(x, i, A) = take(i - 1, A) ⫲ [x] ⫲ drop(i -1, A), if $1 \le i \le \#A < M$.

3.3.2 Deletion of an element

Deletion can be take place from the following positions.

(i) from the beginning of an array
(ii) from the end of an array
(iii) from any position of an array

(i) From the beginning of the array:

$$\text{B-DELETE}(A) = \begin{cases} \text{init}(A), & \text{if } 1 \le \#A; \\ [\,], & \text{if } \#A = 0. \end{cases}$$

(ii) From the end of the array:

$$\text{E-DELETE}(A) = \text{tl}(A), \quad \text{if } \#A > 0.$$

(iii) From any position of the array:

$$\text{DELETE}(i, A) = \text{take}(i-1, A) \Vert \text{drop}(i, A), \quad \text{if } i \ge 1;$$

where i is the location or position in the array.

3.3.3 Retrieval of an element from an array

To retrieve an element from an array we can use the GET operation. This operator can be implemented in the following cases.

(i) from the beginning of a array
(ii) from the end of an array
(iii) from any position of an array

(i) From the beginning of an array:

$$\text{B-GET}(A) = \text{hd}(A), \quad \text{if } \#A > 0.$$

(ii) From the end of an array:

$$\text{E-GET}(A) = \text{last}(A), \text{ if } \#A > 0 \text{ and } \#A - 1 < M.$$

(iii) From any position of an array:

$$\text{GET}(i, A) = \text{hd}(\text{take}(i, A) - \text{take}(i-1, A)), \quad \text{if } 0 < i < \#A.$$

3.3.4 To find the next element of an array at the position i

$$\text{NEXT}(i, A) = \text{hd}(\text{take}(i+1, A) - \text{take}(i, A)), \quad \text{if } 0 \le i \le \#A.$$

3.3.5 To find the previous element of an array at the position i

$$\text{PREVIOUS}(i, A) = \text{hd}(\text{take}(i-1, A) - \text{take}(i-2, A)), \quad \text{if } 2 \le i < \#A.$$

Example 3.3.1: Consider the array of numbers A, given by

$$A = [1, 5, 3, 1, 6, 8, 9]. \text{ Here } \#A = 7.$$

Suppose the maximum array size is 10. Now we use the above operations for this array, i.e. INSERT, DELETE and GET.

3.3.1.1 Insertion Operation

Suppose we want to insert an element x = 15 in the array.

(i) Insertion at the beginning:

$$\text{B-INSERT}(15, A) = \text{cons}(15, A)$$
$$= [15, 1, 5, 3, 1, 6, 8, 9]$$

(ii) Insertion at the end:

$$\text{E-INSERT}(15, A) = A \Vert [15]$$
$$= [1, 5, 3, 1, 6, 8, 9] \Vert [15]$$
$$= [1, 5, 3, 1, 6, 8, 9, 15]$$

(iii) Insertion at any position:

Suppose i = 4, and x = 15.

INSERT(15, 4, A) = take(4 - 1, A) ⫲ [15] ⫲ drop(4 -1, A)

= take(3, A) ⫲ [15] ⫲ drop(3, A)

= [1, 5, 3] ⫲ [15] ⫲ [1, 6, 8, 9]

= [1, 5, 3, 15, 1, 6, 8, 9]

(iv) To find the next and previous element at the position i = 4:

NEXT(4, A) = hd(take(5, A) – take (4, A))

= hd([1, 5, 3, 1, 6] – [1, 5, 3, 1])

= hd([6]) = 6

PREVIOUS(4, A) = hd(take(3, A) – take (2, A))

= hd([1, 5, 3] – [1, 5])

= hd([3]) = 3

3.3.1.2 Deletion Operation

(i) Deletion from the beginning:

B-DELETE(A) = tl(A)

= tl [1, 5, 3, 1, 6, 8, 9]

= [5, 3, 1, 6, 8, 9]

(ii) Deletion from the end:

E-DELETE(A) = init(A)

= init [1, 5, 3, 1, 6, 8, 9]

= [1, 5, 3, 1, 6, 8]

(iii) Deletion from any position:

Suppose we want to delete an element from the position i = 3.

DELETE(3, A) = take(2, A) ⫲ drop(3, A)

= [1, 5] ⫲ [1, 6, 8, 9]

= [1, 5, 1, 6, 8, 9]

3.3.1.3 GET Operation

(i) To get an element from the beginning:

B-GET(A) = hd(A) = 1.

(ii) To get an element from the end:

E-GET(A) = last(A) = 9.

(iii) To get an element from any position:

Suppose the position of the element i = 5.

GET(5, A) = hd(take(5, A) – take (4, A))

$$= hd([1, 5, 3, 1, 6] - [1, 5, 3, 1])$$

$$= hd([6]) = 6.$$

4. FUZZY DATA STRUCTURES THROUGH FUZZY LISTS

The concepts of fuzzy sets and fuzzy bags were introduced and studied by Zadeh [11] and Yager [10] respectively. These notions were extended to define the notion of fuzzy lists in [5] as follows:

Definition 4.1: A fuzzy list L is associated with a positional function P_L, which is defined as

$$P_L : X \times I \rightarrow P(N),$$

where P(N) is the power set of the set of nonnegative integers N.

Thus, for any $x \in X$ and $\alpha \in I$, $P_L(x, \alpha)$ provides the set of positions in which the element x occurs in L with grade of membership α and I = [0, 1].

A *fuzzy stack* (F-STACK) is a linearly ordered data structure which works on the LIFO principle. The membership values are attached with every element in it. However, the operations of F-PUSH, F-POP, F-CHANGE, F-TOP, F-EMPTY and F-PEEP can be defined in similar manner, as we have done for discrete STACKS using list theoretic approach. For example, we can define F-PUSH as follows:

Definition 4.2: Suppose S is an F-Stack. Consider that the F-Stack S contains maximum number of elements M, where M > 0 and S contains #S number of elements. Suppose (x, α) be an element used in the operation, where $\alpha \in [0,1]$. Insertion or addition of an element in S is:

$$\text{F-PUSH}((x, \alpha), S) = \begin{cases} S \Vdash [(x, \alpha)], & \text{if } \#S < M; \\ \text{Overflow}, & \text{if } \#S = M. \end{cases}$$

A *fuzzy queue* (F-QUEUE) is a linearly ordered data structure which works on the FIFO principle. As like fuzzy stack, membership values are attached with every element in it. The operations like FQ-INSERT, FQ-DELETE, FQ-FRONT, FQ-REAR and FQ-EMPTY can be defined as in the corresponding crisp cases. For example, FQ-INSERT can be defined as follows:

Definition 4.3: Suppose Q is an F-Queue. Consider that the Q has maximum size M, where M > 0 and Q contains #Q number of elements. Suppose (x, α) be a element used in the operation. The insertion of an element in Q is:

$$\text{FQ-INSERT}((x, \alpha), Q) = \begin{cases} Q \Vdash [(x. \alpha)] & \text{if } \#Q < M \\ \text{Overflow} & \text{if } \#Q \geq M \textit{ or } Q \text{ size is fixed.} \end{cases}$$

A *fuzzy array* (F-ARRAY) is a finite fuzzy list. So, most of the operations on fuzzy array can be directly defined through operations on fuzzy list. These include insertion of an element (at the beginning, end or any position), retrieve an element and previous element of a particular position. Since these are similar to those for crisp array defined earlier, we outline only one of them for illustration; the FB-INSERT operation.

Definition 4.4: Suppose A is an F-Array with maximum number of elements it can store being M. The number of elements in the array A in its current position is #A. Suppose (x, α) is an element to be added to the list A at the beginning of the array: We define this as

$$\text{FB-INSERT}((x,\alpha), A) = \text{cons}((x,\alpha), A), \text{ if } \#A < M.$$

5. INTUITIONISTIC FUZZY DATA STRUCTURES THROUGH INTUITIONISTIC FUZZY LISTS

The notion of intuitionistic fuzzy list was introduced in [6] is defined as follows:

Definition 5.1: An *intuitionistic fuzzy list* L on a set X is characterized by its position function P_L defined as

$$P_L : X \times J \to P(N)$$

where $J = \{ (\alpha, \beta) : \alpha, \beta \in [0, 1] \text{ and } 0 \leq \alpha + \beta \leq 1 \}$ and P(N) is the set of all subsets of N.

Thus, for any $x \in X$ and $(\alpha,\beta) \in J$, $P_L(x, (\alpha,\beta))$ provides the set of positions in which the element x occurs in L with grade of membership (α,β).

An intuitionistic fuzzy list is an extension of a fuzzy list. The intuitionistic fuzzy data structures like *intuitionistic fuzzy STACK* (IF-STACK), *intuitionistic fuzzy QUEUE* (IF-QUEUE) and *intuitionistic fuzzy ARRAY* (IF-ARRAY) are the extension of their corresponding fuzzy versions.

Various operations on STACK/F-STACK, QUEUE/F-QUEUE and ARRAY/F-ARRAY can be extended to define them on IF-STACK, IF-QUEUE and IF-ARRAY in a natural way. So, we omit them.

Note 5.1: One finds by going through the literature that the notions of fuzzy list and hence those of fuzzy stack, fuzzy queue, fuzzy array and the corresponding intuitionistic fuzzy operations were completely new. We have introduced an approach so as to develop a list theoretic relational database in [9]. Using the notions of fuzzy lists and intuitionistic fuzzy lists and data structures defined based upon them like the array, one can develop a fuzzy list theoretic and intuitionistic fuzzy list theoretic databases. In the modern applications uncertainty has become an integral part and these proposed databases shall be of utmost importance from the application point of view. Some applications of fuzzy lists in playlist management have already been made by Deliege et al [3].

6. CONCLUSION

The definition of lists using the notion of position function as defined by Tripathy et al in 2001 has been further extended to define the notions of Fuzzy lists [4] and intuitionistic fuzzy lists [5]. Application of this new approach has made it simple to define these extended notions and define their properties. Recently one application of these new approaches has been made to define the relational model. In this paper we provided the realization of data structures like queue, stack, and array using the modified definition. Also, we extended these data structures to define their fuzzy as well as intuitionistic fuzzy versions. Many other data structures can also be implemented in a similar manner. A package on lists and operations on them can be developed and then used for developing the above data structures following the methods provided in this article. The applications of Fuzzy and Intuitionistic Fuzzy data structures can be practically implemented in job scheduling in operating systems, memory allocations, and similar types of some real life operations, which we propose as a direction for further research.

7. REFERENCES

[1] Atanassov, K.T.: Intuitionistic Fuzzy Sets, Fuzzy Sets and Systems, 20, (1986), pp.87-96.

[2] Bird, R. and Walder, P.: Introduction to Functional Programming, Prentice Hall International Series in Comp. Sc. (1988).

[3] Deliege, F. and Pedersen, T.B.: Using Fuzzy Lists for Playlist Management, MMM 2008, LNCS 4903, (2008), pp.198-209.

[4] Tripathy, B.K., Jena, S.P. and Ghosh, S.K.: On the Theory of Bags and Lists, Information Sciences (USA), 132(2001), pp. 241-254.

[5] Tripathy, B.K., Jena S.P. and Ghosh, S.K.: On the Theory of Fuzzy Bags and Fuzzy Lists, Int. J. Fuzzy Maths., 9(4), (2001), pp.1209-1220.

[6] Tripathy, B.K. and Choudhury, P.K.: Intuitionistic Fuzzy Lists, Notes On Intuitionistic Fuzzy Sets, Vol. 9, No.2 (2003), pp.61-73

[7] Tripathy, B.K. and Gantayat, S.S.: Some More Properties of Lists and Fuzzy Lists, Information Sciences (USA), 166 (2004), pp.167-179.

[8] Tripathy, B.K. and Pattnaik, G.P.: On Some Properties of Lists and Fuzzy Lists, Information Sciences (USA), 168 (2004), pp.9-23.

[9] Tripathy, B.K. and Gantayat, S.S.: Some new properties of lists and a framework of a list theoretic relational model, Communicated to ACM conference, Coimbatore, (2012).

[10] Yager, R.R.: On the Theory of Bags, Intl. Jour. of General Systems, 13 (1986), pp.23-37.

[11] Zadeh, L.A.: Fuzzy Sets, Information and Control, 8, (1965), pp.338-353.

Authors

Dr. B.K Tripathy is a Senior Professor in the School of Computing Sciences and Engineering, VIT University, at Vellore, Tamil Nadu, India, has published more than 155 technical papers in International journals/ Proceedings of International Conferences/ edited book chapters of reputed publications like Springer and guided 12 students for Ph.D. so far. He is having more than 30 years of teaching experience. Dr. Tripathy is a member of international professional associations like IEEE, ACM, IRSS, CSI, IMS, OITS, OMS, IACSIT, IST and is a reviewer of around 21 international journals which include IEEE, World Scientific, Springer and Science Direct publications. Also, he is in the editorial board of at least 11 international journals. His current research interest includes Fuzzy Sets and Systems, Rough Sets and Knowledge Engineering, Granular Computing, Soft Computing, Data Clustering, Database Anonymisation Techniques, Bag Theory, List Theory and Social Network Analysis.

Dr. S. S. Gantayat is now working as an Associate Professor in the Department of Computing Sciences & Engineering, GMR Institute of Technology, Rajam, Andhra Pradesh, India. He has completed his Ph.D. in Computer Science from Department of Computer Science at Berhampur University, Berhampur, Odisha. He has more than 19 years of teaching and research experience. He has more than 8 research publications in National and International Journals and conferences. He is a member of and life member of IE(I), CSI, ISTE, AIRCC, OITS, IACSIT, IRSS, ISIAM, IMS and SSI. He is a reviewer of 3 international journals and conferences and editorial board member of an American journal in Computer Science. He has the research interests in the fields of Fuzzy Sets and Systems, Rough Set Theory and Knowledge Representation, List Theory and Applications, Soft Computing, and Cryptography.