Bidhan Arya
UG19, 190238
IITK

MULTI ARRAY

INTRODUCTION

When we study about data structures, the first data structures that we encounter are arrays and linked list. Both of these data structures have their pros and cons. The concept of Multi Array tries to take the benefit of the perks of both these data structures. In this paper we discuss the working of Multi Array Data Structure. The table of contents are as follows:

Let us study these in detail.

1. ARRAY PROPERTIES

Arrays are the data structure which store the data as contiguous memory blocks. Due to this reason, an array has to know the size beforehand so that it can reserve the required space. Due to presence of data in contiguous manner, arrays have faster iteration when compared to dynamically alloted linked list. However, this also presents with some problems. Due to fixed size requirement, arrays have fixed size and when they are full, addition of elements require the creation of a new, larger array. Arrays can store only one type of data.

Bidhan Arya
UG19, 190238
IITK

Also, the action of inserting an element in an array is in the order of O(n), because the entire array needs to be copied to another array to accomodate the given element at the required place. Also to delete an element, the elements need to be shifted and hence the order is O(n).

2. LINKED LIST PROPERTIES

Linked lists store data dynamically. Due to this, their size is not fixed and hence can be extended in O(1) time. Also, for deletion or insertion, if the element is known, it can be deleted in contanst time. Insertion follows the same logic. However, randomly accessing data in Linked List is of order O(n) as they do not have indexing. However, they are very effective when the element needs to be inserted near the end or the beginning. The iteration in linked list is slower than that in array, because of non-contiguous nature of storing data.

3. MULTI ARRAY PROPERTIES

Multi array can store any type of data as per the implementation. In the following implementation, we store four types of data, int, float, char and String. The addition, insertion and deletion in multi array is like addition, insertion and deletion in linked list. The iteration in multi array is like that in arrays. That is, they iterate with time complexity of O(1). Since it acts like an array, random access is possible, but also the size of array is fixed, which can be automatically enlarged in implementation. Before iteration, after all the additions and deletions have been done, the linked list backing the array needs to be converted to an array, and so in worst case, if one needs to iterate over the array after each addition or deletion, then the multi array acts like an array. But in the average case, it performs much better than an array.

Bidhan Arya
UG19, 190238
IITK

4. SHORTCOMING OF MULTI ARRAY

Despite the ability of multi arrays to take the best of both arrays and linked list, it does present some shortcomings. Since data types of any type can be stored, while retrieving, the data type should be known to the user, because when the get() method returns the variable, it expects the variable type used to store the returned data to match the data type of the returned variable, and if this is not the case, compiler may throw a runtime error.

However this can be avoided by using Generics in Java. Another problem that arises is the space. Since this data structure adds with the rate similar to that of a linked list, it doesn't delete the array elements, just moves their references, and all the delete commands result in wastage of Data. If space is not a problem, then multi array acts better than both arrays and linked list.

5. UNDERSTANDING THE WORKING OF MULTI ARRAY

Multi array uses arrays and linked list as the backing data structure, and builds upon them. For every data type, an array is created. These arrays are used to store data, and hence their size is prefixed, which can be set to automatically enlarge when required. All of these arrays are connected through a linked list. The linked list is a list of IndexNodes. The IndexNode calss consists of two fields. A character and an integer. The character is used to differentiate the different data types, and their corresponding arrays, and the integer is used to access the elements from the respective array. While adding a new element, the element is added to the

corresponding array, and its index is stored in the Indexnode, which gets connected by a linked list. Lets take an example to understand this better.

Bidhan Arya
UG19, 190238
IITK

We take only two data types for this demonstration, an int and a float. We have two arrays, intArray, and floatArray. Also, we have a node class, and a linked list.

Suppose we add 1 to the array, now

intArray = {1}, floatArray = {}, linkedList = { (‘i’ , 0)}

Next we add 23 to the array, now

intArray = {1,23}, floatArray = {}, linkedList = { (‘i’ , 0) --> (‘i’ , 1)}

Next, we add 2.3f to the array, now

intArray = {1,23}, floatArray = {2.3}, linkedList = { (‘i’ , 0) --> (‘i’ , 1) --> (‘f’ , 0)}

Next, we add 4 to the array, now

intArray = {1,23,4}, floatArray = {2.3},

linkedList = { (‘i’ , 0) --> (‘i’ , 1) --> (‘f’ , 0) --> (‘i’ , 2)}

And so on.

Similarly, when an element is inserted, the same steps are followed. The element is added to the end of the respective array and the node is inserted at the required position, for example,

We insert 43.5f at index 2, now

intArray = {1,23,4}, floatArray = {2.3,43.5},

linkedList = { (‘i’ , 0) --> (‘i’ , 1) --> (‘f’ , 1) -- > (‘f’ , 0) --> (‘i’ , 2)}

Since the number is always inserted at the end of the array, the order of this operation is O(1), also the order of inserting at the front or end of a linked list is O(1), and for somewhere in between, for average case, is much better than O(n).

For deleting an element, one simply needs to remove the required node, and no changes in the array is required, and hence, the deletion is of the order same as that of a linked list. For example, we delete the element at index 3, now,

Bidhan Arya
UG19, 190238
IITK

intArray = {1,23,4}, floatArray = {2.3,43.5},

linkedList = { (‘i’ , 0) --> (‘i’ , 1) --> (‘f’ , 1) -- > (‘i’ , 2)}

These are the methods to add, insert and delete data from the data structure. Some other methods are created to access the data, and print the array, etc.
After all the data manipulations, the linked list needs to be converted to an array for faster access to data, This operation is of the order O(n), and should be done only when required to access data. Otherwise, if used everytime a data element is added or removed, the multi array acts like an array.

6. CODE

We first create a class for out Multi Array

```
public class MultiArray {


    private int[] intArray = new int[10];
    private float[] floatArray = new float[10];
    private char[] charArray = new char[10];
    private String[] stringArray = new String[10];
    private int intIndex, floatIndex, charIndex,stringIndex;
    private IndexNode[] nodeArray = new IndexNode[30];
```

In the given code, four types of data are required, hence we create four arrays of four types, and the four indexes count the next available index free for element insertion in the respective arrays.

Next, we create a class for IndexNode, and put two fields, one character for arrayType and one index to know the element position in the array. It also has references to two nodes, one previous to this current node and one to the next node. We create a doubly linked list. Also the getters and setters are created.

```
public class IndexNode {
    private int index;
    private char arrayType;
    IndexNode prev;
    IndexNode next;


public IndexNode(int index, char arrayType) {
```

Bidhan Arya
UG19, 190238
IITK

```
this.index = index;
this.arrayType = arrayType;
    }

}
```

We create a IndexList class which is a linkedlist of the IndexNodes. This has three fields, a head node,a tail node and a size integer.

```
public class Indexlist {
    IndexNode head;
    IndexNode tail;
    private int size;

}
```

I.Addition

To add a data element, we first need to create a node, and then add that node to the linked list, and insert the element in respective array. For this we use method overloading in the MultiArray class, for taking in different types of inputs. Then the element is inserted and node is added to the IndexList. The add() method adds to the end of multiarray. To insert anywhere in between, we use the insert() method, which is discussed later.

So, first we create a method in MultiArray class.

```
public void add(int i){
    try {
    intArray[intIndex] = i;
    }
    catch (Exception e){
int[] newIntArray = new int[intArray.length * 2];
System.arraycopy(intArray,0,newIntArray,0,intArray.length);
    intArray = newIntArray;
    intArray[intIndex] = i;
    }
```

```
IndexNode node = new IndexNode(intIndex,'i');
list.add(node);
intIndex++;
}
```

In the above method, we try to add an int to the array if the array is not full, else we first enlarge the array and then add the element, and then we create a node, and

Bidhan Arya
UG19, 190238
IITK

add it to the IndexList. And finally increase the intIndex. Next, we create a method in the IndexList class to add the node.

```
public void add(IndexNode node){
if(head == null){
head = node;
tail = node;
size++;
return;
}
tail.setNext(node);
node.setPrev(tail);
tail = node;
size++;
}
```

Similarly methods can be created for other data types in the MultiArray class, there is no need to create custom function in the IndexList because it just takes in a node.

II. Insertion

To insert the element, we follow a similar procedure. We take the index and the element to be inserted, then the element is added to the respective array at the end. A node is created and this node is added to the index required in the IndexList. First a method is created in the MultiArray Class to take the element to be inserted, here we take demonstration only for integer.

```
public void insert(int index, int i){
if(index>=list.getSize() || list.getSize() == 0){
System.out.println("Invalid function used!!");
return;
}
try {
intArray[intIndex] = i;
}
catch (Exception e){
int[] newIntArray = new int[intArray.length * 2];
```

```
System.arraycopy(intArray,0,newIntArray,0,intArray.length);
intArray = newIntArray;
intArray[intIndex] = i;
}
IndexNode node = new IndexNode(intIndex, 'i');
list.insert(index,node);
intIndex++;


}
```

Bidhan Arya
UG19, 190238
IITK

When the node is created, the node is sent to IndexList to be added. The method in the IndexList class takes two inputs, the element to be inserted and the index where it needs to be added.

```
public void insert(int index, IndexNode node){
IndexNode current = head;
int count = 0;
while(count != index){
current = current.getNext();
count++;
}
if(current != head) {
current.getPrev().setNext(node);
node.setPrev(current.getPrev());
current.setPrev(node);
node.setNext(current);
}
else{
current.setPrev(node);
node.setNext(current);
head = node;
}
size++;
}
```

Similarly, other data types can be inserted in the multi array, using method overload.

III. Deletion

To delete the element, one simply needs to delete the node in the IndexList, without removing any element from the array, and hence the order of removing an element is of the order of linkedList (Indexlist). And for that we create a method in the IndexList code.

```
public void delete(int index){
   IndexNode current = head;
        int count = 0;
      while(count < index){
            count++;
    current = current.getNext();}
```

Bidhan Arya
UG19, 190238,
IITK

```
      if(index == 0){
    head = current.getNext();
      current.setNext(null);
            size--;
              }
   else if(index == size -1){
     tail = current.getPrev();
      current.setPrev(null);
            size--;
           }else{
current.getPrev().setNext(current.getNext());
current.getNext().setPrev(current.getPrev());
            size--;
              }
             }
```

This simply deletes an element at a given index, but since the user never communicates with the linkedList, a method needs to be created in the MultiArray class, which forwards the index to the linked list, after conforming that the index requested to be deleted is valid.

```
          public void delete(int index){
    if(index >= list.getSize() || list.getSize() == 0){
      System.out.println("List index out of bounds");
                return;
                  }
            list.delete(index);
                  }
```

IV. Accessing Element

To access element from the multiarray, a mentod called get() is required, and since the element returned can be of different types, generic data type is used to return the required data element.

This method get() is created in the MultiArray class.

```
public <T> T get(int i){
convert();
char c = nodeArray[i].getArrayType();
if(c == 'i'){
```

Bidhan Arya
UG19, 190238,
IITK

```
return (T) new Integer(intArray[nodeArray[i].getIndex()]);


}
else if(c == 'f'){
return (T) new Float(floatArray[nodeArray[i].getIndex()]);
}
else if(c == 'c'){
return (T) new Character(charArray[nodeArray[i].getIndex()]);
}else{
return (T) new String(stringArray[nodeArray[i].getIndex()]);
}
}
```

V. Printing

To print the entire array, a printMultiArray method is made in the MultiArray class.

```
public void printMultiArray(){
IndexNode current = list.head;
while(current != null){
char t = current.getArrayType();
if(t == 'i'){
System.out.println(intArray[current.getIndex()]);
}
else if(t == 'f'){
System.out.println(floatArray[current.getIndex()]);
}
else if(t == 'c'){
System.out.println(charArray[current.getIndex()]);
}
else if(t == 's'){
System.out.println(stringArray[current.getIndex()]);
}
current = current.getNext();
}
}
```

Also, after the IndexList is converted to nodeArray, a method printArray() is created to print the nodeArray elements. This method is also created in the MultiArray class.

```
public void printArray(){
convert();
System.out.print("[ ");
for(int i=0; i<nodeArray.length && nodeArray[i] != null; i++){
```

Bidhan Arya
UG19, 190238
IITK

```
if(nodeArray[i].getArrayType() == 'i'){


System.out.print(intArray[nodeArray[i].getIndex()] + " ");
}
else if(nodeArray[i].getArrayType() == 'f'){
System.out.print(floatArray[nodeArray[i].getIndex()] + " ");
}else if(nodeArray[i].getArrayType() == 'c'){
System.out.print(charArray[nodeArray[i].getIndex()] + " ");
}else if(nodeArray[i].getArrayType() == 's'){
System.out.print(stringArray[nodeArray[i].getIndex()] + " ");
}
}
System.out.print("]");
}
```

VI. Converting

To convert the IndexList containing all the nodes, to a nodeArray, for faster access, a method is created in the MultiArray class. This method, called convert(), should be used before any iteration of the array, or errors may occur.

```
public void convert(){
int count=0;
IndexNode current = list.head;
while(current !=null){
try {
nodeArray[count] = current;
}
catch (Exception e){
IndexNode[] newNodeArray = new IndexNode[nodeArray.length * 2];
System.arraycopy(nodeArray,0,newNodeArray,0,intArray.length);
nodeArray = newNodeArray;
nodeArray[count] = current;
}
count++;
current = current.getNext();
```

```
    }
}
```

Bidhan Arya
UG19, 190238
IITK

7. CONCLUSION

MultiArray can be used to create a dynamic array, which can store different types of data, without compromising speed, however it comes with some problems of space and data type errors, and hence, should be used carefully.

THANK YOU...