Data Structures

Linear List Array Representation

Andres Mendez-Vazquez

August 15, 2016

Outline

Outline

1. Linear List Array Representation
   - The ADT for the list
   - Simplicity After All
   - Operations in Array List
     - Add in Array List
     - Get in Array Linear List
     - By the way the rest is for you to implement
2. Dynamic Arrays
   - The Common Problem
   - How much we need to expand it...
   - A Strategy for it
   - Analysis

How do we implement the ADT List?

In our first representation will use an array

Use a one-dimensional array **element**[]:

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

The previous array

A representation of $L = (a, b, c, d, e)$ using position i in **element**[i].

How do we implement the ADT List?

In our first representation will use an array

Use a one-dimensional array **element**[]:

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

The previous array

A representation of $L = (a, b, c, d, e)$ using position i in **element**$[i]$.

Where to map in the array

Right To Left Mapping

-	-	-	-	e	d	c	b	a

Mapping That Skips Every Other Position

a	-	b	-	c	-	d	-	e	-	-

Wrap Around Mapping

d	e	-	-	-	-	-	-	a	b	c

Where to map in the array

Right To Left Mapping

-	-	-	-	e	d	c	b	a

Mapping That Skips Every Other Position

a	-	b	-	c	-	d	-	e	-	-

Wrap Around Mapping

d	e	-	-	-	-	-	-	a	b	c

Where to map in the array

Right To Left Mapping

-	-	-	-	e	d	c	b	a

Mapping That Skips Every Other Position

a	-	b	-	c	-	d	-	e	-	-

Wrap Around Mapping

d	e	-	-	-	-	-	-	a	b	c

Outline

Representation Used In Text

Something Notable

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

Size=5

Thus

1. Put element i of list in **element**$[i]$.
2. Use a variable size to record current number of elements

Representation Used In Text

Something Notable

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

Size=5

Thus

1. Put element i of list in **element**$[i]$.
2. Use a variable size to record current number of elements

Code

An Implementation

```
public class SimpleArrayList <Item> implements
                                      LinearList <Item>{
// private elements of implementation
protected Item element[];
protected int size;
protected final static int DEFAULT_SIZE = 10;

//Constructors
public SimpleArrayList(){
    this.size = 0;
    this.element = (Item[]) new Object[this.DEFAULT_SIZE];
}

public SimpleArrayList(int NewSize){
    this.size = 0;
    this.element = (Item[]) new Object[NewSize];
}
```

Data Type Of Array element[]

First than anything

Data type of list elements is unknown.

Data Type Of Array element[]

First than anything

Data type of list elements is unknown.

Thus, we used the genericity of Object

1. Then, we use element[] to be of data type Object.
2. Then, we cast to the new "Item."

Data Type Of Array element[]

First than anything

Data type of list elements is unknown.

Thus, we used the genericity of Object

1. Then, we use element[] to be of data type Object.
2. Then, we cast to the new "Item."

However

You cannot put elements of primitive data types (**int, float, double, char,** etc.) into our linear lists.

Data Type Of Array element[]

First than anything

Data type of list elements is unknown.

Thus, we used the genericity of Object

1. Then, we use element[] to be of data type Object.
2. Then, we cast to the new "Item."

However

You cannot put elements of primitive data types (**int, float, double, char,** etc.) into our linear lists.

Outline

Outline

Operations: Add

Add/Remove an element

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

Size=5

add(1,g)

a	g	b	c	d	e	-	-	-
0	1	2	3	4	5	6	7	8

Size=6

Operations: Add

Add/Remove an element

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

Size=5

add(1,g)

a	g	b	c	d	e	-	-	-
0	1	2	3	4	5	6	7	8

Size=6

Code

An Implementation

```
public void add(int index,Item myobject){
   // Initial Variables
   int i;
   // Always check for possible errors
   if (this.size == element.lenght){
      System.out.println("List␣does␣not␣have␣space");
      System.exit(0);
   }
   if (index<0 || index>this.size){
      System.out.println("Index␣out␣of␣bound");
      System.exit(0);
   }
   // Shift postiions as necessary
   for (i = this.size; i>index; i--)
            element[i+1]=element[i];
   //copy element into container
   element[i+1]=myobject;
}
```

Outline

Operations: Get

Here

We have

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

Size=5

get(3)

It will return "d".

The code is simple

Operations: Get

Here

We have

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

Size=5

get(3)

It will return “d”.

The code is simple

Operations: Get

Here

We have

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

Size=5

get(3)

It will return “d”.

The code is simple

```
public Item get(int index){
  //Check always
  if (this.size == 0) return null;
  if (index<0 || index>this.size -1){
      System.out.println("Index out of bound");
      System.exit(0);
   }
   return element[index]
}
```

Outline

You can implement the rest

Yes

Part of your homework!!!

Outline

Now, we have a common problem

Because

We do not know how many elements will be stored at the list.

We use

An initial length and dynamically increase the size as needed.

Now, we have a common problem

Because

We do not know how many elements will be stored at the list.

We use

An initial length and dynamically increase the size as needed.

Example

Example

Length of array element[] is 6:

a	b	c	d	e	f

First create a new and larger array

NewArray = (Item[]) new Object[12]

-	-	-	-	-	-	-	-	-	-	-	-

Now copy the new elements into the new array!!!

System.arraycopy(element, 0, newArray, 0, element.length);

a	b	c	d	e	f	-	-	-	-	-	-

Example

Example

Length of array element[] is 6:

a	b	c	d	e	f

First create a new and larger array

NewArray = (Item[]) new Object[12]

-	-	-	-	-	-	-	-	-	-	-	-

Now copy the new elements into the new array!!!

System.arraycopy(element, 0, newArray, 0, element.length);

a	b	c	d	e	f	-	-	-	-	-	-

Example

Example

Length of array element[] is 6:

a	b	c	d	e	f

First create a new and larger array

NewArray = (Item[]) new Object[12]

-	-	-	-	-	-	-	-	-	-	-	-

Now copy the new elements into the new array!!!

System.arraycopy(element, 0, newArray, 0, element.length);

a	b	c	d	e	f	-	-	-	-	-	-

Example

Finally, rename new array

- element = NewArray;

element[0] ⟶

a	b	c	d	e	f	-	-	-	-	-	-

Outline

First Attempt

What if you use the following policy?

At least 1 more than current array length.

First Attempt

What if you use the following policy?

At least 1 more than current array length.

Thus

What would be the cost of all operations?

- Generate a new array.
- Copy all the items to it.
- insert the new element at the end.

First Attempt

What if you use the following policy?

At least 1 more than current array length.

Thus

What would be the cost of all operations?

- Generate a new array.
- Copy all the items to it.
- insert the new element at the end.

First Attempt

What if you use the following policy?

At least 1 more than current array length.

Thus

What would be the cost of all operations?

- Generate a new array.
- Copy all the items to it.
- insert the new element at the end.

First Attempt

What if you use the following policy?

At least 1 more than current array length.

Thus

What would be the cost of all operations?

- Generate a new array.
- Copy all the items to it.
- insert the new element at the end.

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

Thus, we have

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

Thus, we have

Ok

Not a good idea!!!

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

Thus, we have

Ok

Not a good idea!!!

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

Thus, we have

$$1 + 2 + 3 + \cdots + n = \frac{n\,(n+1)}{2} = O\left(n^2\right) \tag{1}$$

Ok

Not a good idea!!!

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

Thus, we have

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = O\left(n^2\right) \tag{1}$$

Ok

Not a good idea!!!

Outline

A better strategy

Dynamic Array

To avoid incurring the cost of re-sizing many times, dynamic arrays re-size by an amount a.

In our example we double the size, $a = 2$

A better strategy

Dynamic Array

To avoid incurring the cost of re-sizing many times, dynamic arrays re-size by an amount a.

In our example we double the size, $a = 2$

```
Item NewArray[];
if (this.size == element.lenght){
    // Resize the capacity
    NewArray = (Item[]) new Object[2*this.size]
    for(int i=0; i < size ; i++){
        NewArray[i]=element[i];
    }
    element = NewArray;
}
```

Space Complexity

Every time an insertion triggers a doubling of the array

Thus, space wasted in the new array:

$$\text{Space Wasted} = \text{Old Lenght} - 1 \tag{2}$$

a	b	c	d	e	f	-	-	-	-

Wasted

Remember: We double the array and insert!!!

Thus, the average space wasted is

$$\Theta(n) \tag{3}$$

Space Complexity

Every time an insertion triggers a doubling of the array

Thus, space wasted in the new array:

$$\text{Space Wasted } = \text{Old Lenght } - 1 \tag{2}$$

a	b	c	d	e	f	-	-	-	-

Wasted

Remember: We double the array and insert!!!

Thus, the average space wasted is

$$\Theta\left(n\right) \tag{3}$$

For example, we have the following

A trade-off between time and space

- You have and average time for insertion is $\frac{2}{2-1}$.
- In addition, an upper bound for the wasted cells in the array is $(2-1)\,n-1=n-1$.

Actually a more general term of expansion, a

Thus, we have:

- Average time for insertion is $\frac{a}{a-1}$.
- An upper bound for the wasted cells in the array is $(a-1)\,n-1=an-n-1$.

Different languages use different values

- Java, $a=\frac{3}{2}$.
- Python, $a=\frac{9}{8}$

For example, we have the following

A trade-off between time and space

- You have and average time for insertion is $\frac{2}{2-1}$.
- In addition, an upper bound for the wasted cells in the array is $(2-1)\,n-1=n-1$.

Actually a more general term of expansion, a

Thus, we have:

- Average time for insertion is $\frac{a}{a-1}$.
- An upper bound for the wasted cells in the array is $(a-1)\,n-1=an-n-1$.

Different languages use different values

- Java, $a=\frac{3}{2}$.
- Python, $a=\frac{9}{8}$.

For example, we have the following

A trade-off between time and space

- You have and average time for insertion is $\frac{2}{2-1}$.
- In addition, an upper bound for the wasted cells in the array is $(2-1)\,n-1=n-1$.

Actually a more general term of expansion, a

Thus, we have:

- Average time for insertion is $\frac{a}{a-1}$.
- An upper bound for the wasted cells in the array is $(a-1)\,n-1=an-n-1$.

Different languages use different values

- Java, $a=\frac{3}{2}$.
- Python, $a=\frac{9}{8}$

Outline

We have the following final analysis

Amortized Analysis Vs. Classic

	Array Classic Analysis	Dynamic Array Amortized Analysis
Indexing	$O(1)$	$O(1)$
Search	$O(n)$	$O(n)$
Add/Remove	$O(n)$	$O(n)$
Space Complexity	$O(n)$	$O(n)$