ECU178 Computer Science: 210CT - Programming, Algorithms and Data Structures Portfolio

Due on Monday, December 15th, 2014

Dr James Shuttleworth

Robert Rigler : 4939377

Contents

Item 1: Week 3 - Linear Search and Duplicate Finder

1. Pseudocode for linear search

This Simple Algorithm demonstrates how to perform a linear search.
Input: This algorithm takes a populated array A and a value to search for v, as parameters.
Output: The Algorithm is a boolean type and returns either True or False respective of whether the v was found in the list or not.

Algorithm 1 LinearSearch

1: **procedure** BOOL LINEARSEARCH(v, $A[\,]$)
2:     **for** each element i in A **do**
3:         **if** $A[i] = v$ **then**
4:             **return** true
5:         **end if**
6:     **end for**
7: **return** false
8: **end procedure**

---

2. Pseudocode for finding duplicates in a list

This algorithm demonstrates how to examine if a list has duplicate entries using a linear search.
Input: This algorithm takes a populated array A as a parameter.
Output: This Algorithm is a boolean type and returns true or false respective of whether a duplicate value is found or not.

Algorithm 2 Examining for duplicates

1: **procedure** BOOL EXFORDUPES($A[\,]$)
2:     **for** each element i in $A[\,]$ **do**
3:         **for** each element j in $A[\,]$ **do**
4:             **if** $A[i] = A[j]$ **then**
5:                 **return** true
6:             **end if**
7:         **end for**
8:     **end for**
9: **end procedure**

Item 2: Week 4 - Time complexities and Big-O notation

1. Describe the runtime bounds of the linear search algorithm

---
Algorithm 3 LinearSearch

1: **procedure** BOOL LINEARSEARCH(*item*, *list*[ ])

2:

3:　　**for** each element *i* in *list* **do**　　　　(n)

4:　　　　**if** *list*[*i*] = *list* **then** t　　　　(n)

5:　　　　　　**return** true　　　　(n)

6:　　　　**end if**

7:　　**end for**

8: **return** false　　　　　　　　(1)

9: **end procedure**

Collecting the line-by-line runtime data from the algorithms gives: $n + n + n + 1$ which is equivalent to: $3n + 1$.

Therefore the time complexity of the algorithm is O(n).

2. Describe the runtime bounds of the duplicate finder algorithm

---
Algorithm 4 Examining for duplicates

1: **procedure** BOOL EXFORDUPES(*list*[ ])

2:　　**for** each element *i* in *list*[ ] **do**　　　　(n)

3:　　　　**for** each element *j* in *list*[ ] **do**　　　　(n*n)

4:　　　　　　**if** *list*[*i*] = *list*[*j*] **then**　　　　(n*n)

5:　　　　　　**return** true　　　　(n*n)

6:　　　　　　**end if**

7:　　　　**end for**

8:　　**end for**

9: **return** false　　　　　　　　(1)

10: **end procedure**

Collecting the line-by-line runtime data from the algorithms gives: $n + (n * n) + (n * n) + (n * n) + 1$ which is equivalent to: $3n^2 + n + 1$.

Therefore the time complexity of the algorithm is O(n^2)

Additional work: Critical values of relative runtimes

Write a function that determines the critical value at which the relative runtime of two linear algorithms swap.

For this algorithm, I am assuming that $k1 > k2$ (Expression 1 >Expression 2, when $n = 0$). The Algorithm is very simple; While the value of Expression 1 is greater than Expression 2, increase the value of n.

When the Runtime of the algorithms swap, the while-loop exit condition is fulfilled and the current value of n is returned.

Algorithm 5 Relative runtime comparison algorithm

1: **procedure** CRITVAL$(m1, k1, m2, k2)$
2:     **while** $(m1 * n + k1) > (m2 * n + k2)$ **do**
3:         $n + +$
4:     **end while**
5: **return** n
6:
7: **end procedure**

Item 3: Week 6 - Harmonic Series

1. Harmonic Series (Pseudocode)

Use pseudocode to specify a recursive algorithm to compute the nth value of the harmonic series, for some integer n.

The Harmonic series is as follows: $1 + 1/2 + 1/3 + 1/4 + 1/5 + ..1/n$
Input: This algorithm takes two parameters t and n which are the total sum of the algorithm and the number of repetitions, respectively.
Output: This algorithm outputs the value t which is the total sum of the harmonic series.

This procedure uses a while-loop to control the number recursive iterations.
While the number of iterations left is above 0, add the next value to t, decrease the number of iterations by 1 and recursively call the procedure with the new values of t and n.
When the number of iterations left is no longer above 0, the final value of t is returned and the procedure ends.

Algorithm 6 Computing nth value of harmonic series

1: **procedure** HARM(float t, float n)
2:     **while** $n > 0$ **do**
3:         $t \leftarrow t + (1/n)$
4:         $n--$
5:         $HARM(t, n)$
6:     **end while**
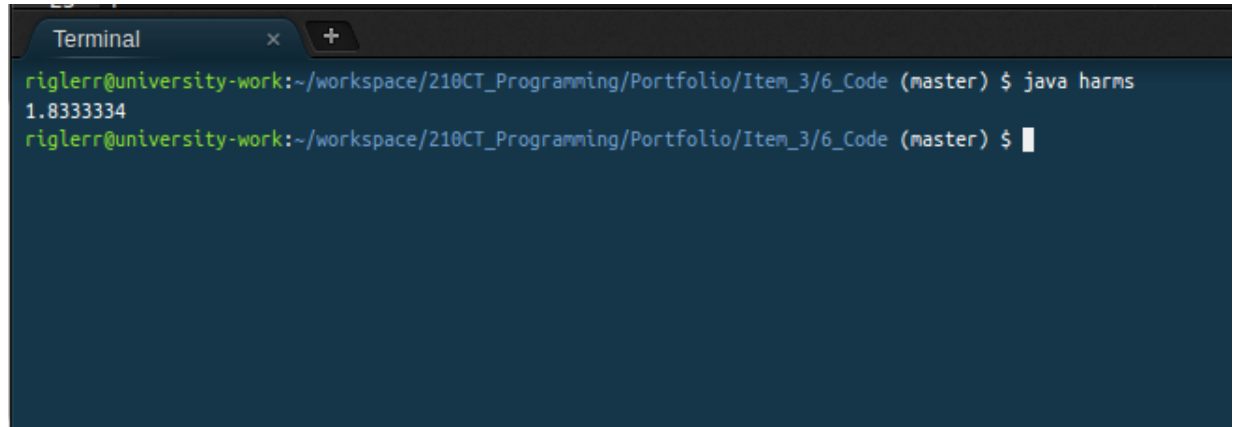7: **return** t
8: **end procedure**

---

2. Harmonic Series (JAVA Implementation)

The Harmonic Series computation algorithm implemented in Java

Listing 1: harms java class file

```java
public class harms{

    public static void main(String[] args){
        /**/
        System.out.println(f(0,3));
    }

    public static float f(float t, float n){
        /*
        t always has a value of 0 on the initial method call.
        n is the nth term, which decreases by 1 each recursive call.

        When n = 0, stop recursive calling and return the value t.
        */
        while (n>0)
        {
            t+= (1/n);
            f(t,--n);
        }
        return t;
    }


}
```

Evidence of the Harmonic Series computation java implementation.
The nth value passed to the method was 3.

Item 4: Week 7 - Heapworksheet or RPN Calculator

Item 5: Week 8 - Linked List Delete function

Below are the Linked List and Node classes that I have created as well as a Test class.
This is a fully functional doubly Linked List capable of Inserting, Prepending, Deleting and
Displaying items in the List. Please see the Evidence Screenshot below:

Listing 2: List Class Implementation: Java

```java
package com.company;


public class List {
    //Nodes to hold the head and tail of the list respectively
    Node head;
    Node tail;
    //------------------------------------------------------------
    //----------List Constructor to init values to null----------
    //------------------------------------------------------------
    public List(){
        head = null;
        tail = null;
    }
    //------------------------------------------------------------
    //-------Insert an Element at an offset from head node--------
    //------------------------------------------------------------

    public void Insert(Node n, Node x){
        //If previous node exists
        if(n != null){
            //sWAP NODE POINTERS
            x.next = n.next;
            n.next = x;
            x.prev =n;

            // If new node isn't end of list
            if(x.next != null)
                x.next.prev =x;
        }
        //Prepend to existing list
        if (head != null & n == null) {
            n = head;
            n.prev = x;
            x.next = n;
            x.prev = null;
            head = x;
        }
```

```java
38
39
40
41          // If new list (First item to be placed)
42        else   if(n == null & head == null)
43          {
44     //make new node equal to both head and tail
45            head = tail =x;
46            x.prev =x.next = null; // make head pointers null
47        } //If all else exp
48        else if(tail == n)
49            tail =x;
50    }
51    //--------------------------------------------------------
52    //--------------DELETE A NODE FROM THE LIST---------------
53    //--------------------------------------------------------
54  public void Delete(Node n){
55      //create temporary node
56      Node i = head;
57
58      while (i != null){ // Loop until null (tail.next)
59
60          if (i.value == n.value){
61
62              if(i == head){ //If node to remove is head node
63                  head = i.next;
64                  i.prev = null;
65                  break;
66              }
67
68        //if node to remove is tail node
69              else if (i == tail){
70                  tail = i.prev;
71                  i.prev.next = null;
72              }
73              else { // If node to remove is not Head/Tail
74              i.prev.next = i.next;
75              i.next.prev = i.prev;
76              break;}
77          }
78          i=i.next; // Increment
79      }
80    }
```

```
81
82
83
84        //----------------------------------------------------
85        //----------DISPLAY ALL NODES IN LIST-----------------
86        //----------------------------------------------------
87     public void display(){
88         Node i = head;
89
90         while(i != null) {
91             System.out.print(i.value + ", ");
92             i = i.next;
93         }
94     }
95
96
97 }
```

Listing 3: Node Class Implementation: Java

```
1  package com.company;
2
3  /**
4   * Created by rob on 20/12/14.
5   */
6
7  public class Node {
8      int value;
9      Node prev;
10     Node next;
11
12     public Node(int val){
13
14             //Node Constructor
15         value = val;
16         next =null;
17         prev =null;
18
19     }
20
21
22 }
```

Listing 4: Test class

```java
package com.company;


public class Main {

    public static void main(String[] args) {
        List obj1 = new List();


        obj1.Insert(null, new Node(42));
        obj1.Insert(obj1.head, new Node(32));
        obj1.Insert(obj1.head.next, new Node(102));


        //----------------------------------------------------
        //---------------Test Inserting Tail------------------
        //----------------------------------------------------
        System.out.println("Insert 3 items:");
        obj1.display();
        System.out.println();


        // ---------------------------------------------------
        //----------------Test Insert Between-----------------
        //----------------------------------------------------
        //Insert 12 between the head node and the head.next
        System.out.println("Insert 12 after head node: ");
        obj1.Insert(obj1.head, new Node(12));
        obj1.display();
        System.out.println();


        //----------------------------------------------------
        //-------------------Test Prepend---------------------
        //----------------------------------------------------
        //Insert 13 at start of pre-existing list
        System.out.println("Prepend 13: ");
           obj1.Insert(null,new Node(13));
           obj1.display();
          System.out.println();
```

```java
43         //-----------------------------------------------------
44         //-----------------Test Delete Head------------------
45         //-----------------------------------------------------
46         //Delete the head node '13' from the list
47         System.out.println("Delete Head: ");
48         obj1.Delete(new Node(13));
49         obj1.display();
50         System.out.println();
51
52         //-----------------------------------------------------
53         //-----------------Test Delete Norm-------------------
54         //-----------------------------------------------------
55         //Delete a middle node (32) from list an display
56         System.out.println("Delete a Middle Node (32)");
57         obj1.Delete(new Node(32));
58         obj1.display();
59         System.out.println();
60
61         //-----------------------------------------------------
62         //----------------Test Delete Tail--------------------
63         //-----------------------------------------------------
64         //Delete the tail node '42' from list an display
65         System.out.println("Delete tail: ");
66         obj1.Delete(new Node(102));
67         obj1.display();
68         System.out.println();
69
70
71     }}
```

```
/usr/lib/jvm/java-8-oracle/bin/java ...
Insert 3 items:
42, 32, 102,
Insert 12 after head node:
42, 12, 32, 102,
Prepend 13:
13, 42, 12, 32, 102,
Delete Head:
42, 12, 32, 102,
Delete a Middle Node (32)
42, 12, 102,
Delete tail:
42, 12,
```