Solution for Midterm Exam 1

PROBLEM 1 (10 points)

“X Matrix” is an nxn sparse matrix in which all non-zero terms are on two diagonals of the square Matrix. That means the x_{i,j} is part of “X Matrix” iff i=j or i=n-j+1. For example, following are a 4x4 “X Matrix” and a 5x5 “X Matrix”.

\[
\begin{array}{cccc}
1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 \\
\end{array}
\]

Assuming that the 2D index of the first element of “X Matrix” starts from (1, 1) and the index of 1D array start from 1. We store an nxn “X Matrix” into a 1D array, in the form as follows:

\[
\begin{array}{cccccccc}
x_{1,1} & x_{1,n} & x_{2,2} & x_{2,n-1} & \cdots & x_{n-1,2} & x_{n-1,n-1} & x_{n,1} & x_{n,n}
\end{array}
\]

a) How many elements in such a 1D array?
b) What is the corresponding 1D index of “X Matrix” element x_{i,j} in the 1D array?

Solution:

a) When n is even, it needs 2n elements. When n is odd, it needs 2n-1 elements.
b) When n is even, the index is

\[
\begin{align*}
2i-1 & \quad \text{when } j \leq n/2 \\
2i & \quad \text{when } j > n/2
\end{align*}
\]

When n is odd, the index is

\[
\begin{align*}
2i-1 & \quad \text{when } i = j \\
2i & \quad \text{when } i = n-j+1
\end{align*}
\]
PROBLEM 2 (20 points)

Given a hash table with 26 buckets (b = 26), and a hash function

\[ F(x) = \text{ord(first character of x)} - \text{ord('A')}. \]  
where \text{ord(x)} means the ASCII value of x.

The ASCII value of each letter from ‘A’ to ‘Z’ increases by 1 for adjacent letters. For example, \( F(\text{Burke}) = \text{ord('B')} - \text{ord('A')} = 1 \). Insert the elements whose keys are: \text{Burke}, \text{Ekers}, \text{Broad}, \text{Blum}, \text{Attlee}, \text{Alton}, \text{Hecht} and \text{Ederly} in this order. Use linear probing to resolve collisions.

a) Draw the hash table following each insertion.

b) What is the loading factor of your table after the last insert?

c) What is the maximum and average number of buckets examined in a successful search of your hash table?

d) What is the maximum and average number of buckets examined in an unsuccessful search of your hash table? (Assuming the hash function is uniform.)

Solution:

a) 

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>⋮</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attlee</td>
<td>Burke</td>
<td>Broad</td>
<td>Blum</td>
<td>Ekers</td>
<td>Alton</td>
<td>Ederly</td>
<td>Hecht</td>
<td>⋯</td>
<td></td>
</tr>
</tbody>
</table>

b) \( \frac{8}{26} \)

c) \text{Attlee 1, Burke1, Broad 2, Blum 3, Ekers 1, Alton 6, Ederly 3, Hecht 1}

   maximum: Alton 6    average: \((1+1+2+3+1+6+3+1)/8=18/8\)

   d) maximum: 9    average: \((9+8+7+6+5+4+3+2+18)/26=62/26\)
PROBLEM 3 (20 points)

Start with an LZW compression dictionary that has the entries (a, 0) and (b, 1)

a) Compress the string sequence "aaabaabab". For each character encountered, draw the code table following the encoding of each character. The initial configuration is shown below. Complete the remaining process.

<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

aaabaabab

Compressed string = null

b) Decompress the code sequence "010242". For each code encountered, draw the decode table following the decoding of each code. The initial configuration is shown below. Complete the remaining process.

<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

010242

Decompressed string = null

Solution

a) Compress:

1: Initial Configuration
<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

aaabaabab

Compressed string = null

2:
<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
<td>aa</td>
</tr>
</tbody>
</table>

aaabaabab

Compressed string = 0
3:

<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
<td>aa</td>
<td>aab</td>
</tr>
</tbody>
</table>

aaabaabab
Compressed string = 02

4:

<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
<td>aa</td>
<td>aab</td>
<td>ba</td>
</tr>
</tbody>
</table>

aaabaabab
Compressed string = 021

5:

<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
<td>aa</td>
<td>aab</td>
<td>ba</td>
<td>aaba</td>
</tr>
</tbody>
</table>

aaabaabab
Compressed string = 0213

6:

<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
<td>aa</td>
<td>aab</td>
<td>ba</td>
<td>aaba</td>
<td>ab</td>
</tr>
</tbody>
</table>

aaabaabab
Compressed string = 02130

7:

<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
<td>aa</td>
<td>aab</td>
<td>ba</td>
<td>aaba</td>
<td>ab</td>
</tr>
</tbody>
</table>

aaabaabab
Compressed string = 021301

b) Decompress:

1: Initial Configuration

<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

010242
Decompressed string = null

2:

<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
<td>ab</td>
</tr>
</tbody>
</table>

010242
Decompressed string = a

3:

<table>
<thead>
<tr>
<th>code</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>a</td>
<td>b</td>
<td>ab</td>
<td>ba</td>
</tr>
</tbody>
</table>

010242
Decompressed string = ab
4:

code 0 1 2 3 4
key a b ab ba aa

010242
Decompressed string = aba

5:

code 0 1 2 3 4 5
key a b ab ba aa aba

010242
Decompressed string = abaab

6:

code 0 1 2 3 4 5 6
key a b ab ba aa aba aaa

010242
Decompressed string = abaabaab

7:

code 0 1 2 3 4 5 6
key a b ab ba aa aba aaa

010242
Decompressed string = abaabaab
PROBLEM 4 (15 points)

Consider the "rat in a maze" problem, depicted below:

<table>
<thead>
<tr>
<th>W</th>
<th>X</th>
<th>X</th>
<th>W</th>
<th>X</th>
<th>X</th>
<th>W</th>
<th></th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>X</td>
<td>W</td>
<td>X</td>
<td>X</td>
<td>W</td>
<td></td>
<td></td>
<td>W</td>
</tr>
<tr>
<td>W</td>
<td>X</td>
<td>W</td>
<td>X</td>
<td>X</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>W</td>
<td>X</td>
<td>W</td>
<td>X</td>
<td>W</td>
<td>W</td>
<td></td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>W</td>
<td>X</td>
<td>X</td>
<td>W</td>
<td>X</td>
<td>X</td>
<td>W</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>W</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>W</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>W</td>
<td>O</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>W</td>
<td>X</td>
<td>X</td>
<td>W</td>
<td>O</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>O</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>O</td>
<td>W</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>O</td>
<td>W</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>W</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>W</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
<td>W</td>
<td>W</td>
<td></td>
</tr>
</tbody>
</table>

a) The start location for the rat is S, and the finish is at F. The locations marked W contain impenetrable walls. Simulate the stack-based search for the Finish. The rat attempts to move up, right, down, and left (in that order!). There’s no need to depict the stack. As you arrive on a square, place a circle in it. As you retreat from a square, add an X to it. You may find it useful to draw a line from the circle to the next, in certain places.
b) Simulate the (queue-based) search for the shortest path from the Start to Finish in the maze. Mark each square visited with the length of the shortest path from the Start to that square. Remember, the rat can only move horizontally and vertically. There is no need to depict the queue.

**What is the length of the shortest path from the Start to Finish?**

Determine the least accessible square, i.e. the square that takes the longest to get to. This is the one whose shortest path from Start is longer than that of any other square.
PROBLEM 5 (15 points)

Assuming that all the nodes in the chain are of type Integer. Let firstNode be the head of the chain. The node of the chain is define as follows:

```java
class ChianNode{
    public Object element;
    public ChainNode next;
}

class Chain{
    protected ChainNode firstNode;
    int MaxKey(ChainNode p);
    int NumberOfNodes(ChainNode p);
    float AverageValue (ChainNode p);
}
```

Please give the recursive algorithms of the following functions:

a) The maximum value of element in the chain MaxKey(firstNode);
b) The number of nodes in the chain NumberOfNodes(firstNode);
c) The average value of all element’s key AverageValue (firstNode).

Solution

a)

```java
int MaxKey(ChainNode p){
    if (p==null) return -1;
    int result= (Integer)(p.element);
    if (p.next==null) return result;
    return result >= MaxKey(p.next) ? result, MaxKey(p.next);
}
```
b) int NumberOfNodes(ChainNode p) {
    if (p==null) return 0;
    if (p.next==null) return 1;
    return 1 + NumberOfNodes(p.next);
}

c) float AverageValue (ChainNode p) {
    if (p==null) return 0;
    int result= (Integer)(p.element);
    if (p.next==null) return result;
    int numberOfNodes= NumberOfNodes(p.next);
    return (result + AverageValue(p.next) * numberOfNodes) / (numberOfNodes + 1);
}
PROBLEM 6 (20 points)

In this problem we are trying to sort the elements in the queue in ascending order. You can only use only one stack to assist the sorting. Other data structures such as arrays or lists are **not** allowed. The template for the class is given below:

```java
package dataStructures;

public class MyClass {
    protected ArrayStack stack;

    // NO other data members allowed

    // constructor
    public MyClass() {
        stack = new ArrayStack();
    }

    /** sort the queue */
    public void sortTheQueue(ArrayQueue que) {
        // your code goes here
    }

    /** test code */
    public static void main(String[] args) {
        ArrayQueue que = new ArrayQueue();
        que.put(new Integer(6));
        que.put(new Integer(5));
        que.put(new Integer(9));
        que.put(new Integer(4));
        que.put(new Integer(3));

        MyClass test = new MyClass();
        test.sortTheQueue(que);
    }
}
```

a) Write Java code for the function `sortTheQueue()` in the class `MyClass`. You can only use the `put()` and `remove()` methods of the `ArrayQueue`. You can use all the methods of the `ArrayStack`.

b) What is the time and space complexity of the method `sortTheQueue()`?
Solution:

a) /* sort the queue */
   
   public void sortTheQueue(ArrayQueue que) {
       Integer temp = (Integer) que.remove();
       if (temp == null)
           return;
       stack.push(temp);
       temp = (Integer) que.remove();
       while (temp != null) {
           while (!stack.empty() && temp > (Integer) stack.peek()) {
               Object topOfStack = stack.pop();
               que.put(topOfStack);
           }
           stack.push(temp);
           temp = (Integer) que.remove();
       }

       while (!stack.empty()) {
           System.out.println(stack.peek().toString());
           que.put(stack.pop());
       }
   }

b) O(n²)