CS 134 Midterm Solutions

Fall 2009

This is a closed book exam. You have 90 minutes to complete the exam. There are 5 questions on this examination. The point values for the questions are shown in the table below. Your answers should fit in the space provided in the exam booklet. Paper for scrap work will be made available during the examination.

Good Luck!

NAME: __________________________________________

<table>
<thead>
<tr>
<th>Question</th>
<th>Point</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

I have neither given nor received aid on this examination.

Signature: ___________________________
1) Suppose that two Ethernet stations named A and B are connected to opposite ends of a maximal length Ethernet, and that a third station C is attached to the network at some other point. Assume that A first sends a complete packet successfully, then B sends a complete packet successfully and finally, C sends a complete packet successfully. While C is finishing its transmission, A and B each decide that they would like to send another packet. In this case, A and B will both wait for C to complete the transmission of its frame. Then, A and B will both transmit and collide with one another as soon as the C’s transmission completes. After this collision, A and B will both choose to delay for a random number of slot times before transmitting again, following the rules of exponential backoff.

a) From what range of delay slots will each station choose before making its second attempt to transmit? What is the probability that A and B will collide during their second attempts to transmit?

Each station will choose between delaying 0 or 1 slot times. There are a total of four equally likely pairs of delay values they might choose --- (0,0), (0,1), (1,0), (1,1). Two of these combinations lead to a second collision so the probability of a collision is 1/2.

b) Suppose that they both choose the same delay during their second attempts to transmit so that they collide a second time. They will again choose random delay times and attempt to transmit after these delay periods are complete. In this case, what range of backoff times will they use and what is the probability that they will collide during their third attempt?

Each station will choose between delaying 0, 1, 2 or 3 slot times. There are a total of 16 equally likely pairs of delay values they might choose. Four of these combinations lead to a second collision so the probability of a collision is 1/4.

c) Suppose that they still happen to choose identical delays so that their third attempts also collide. Finally, in the fourth attempt, assume that they choose different delay periods and station A transmits successfully. The actual amount of time consumed by this entire process depends on the random delays the stations choose. What is the smallest amount of time (measured in slot times) that might elapse from the end of C’s transmission to the beginning of A’s successful transmission? Explain your answer.

Each of the collisions described will consume one slot. In the fourth round, A can transmit immediately so that no additional delay in incurred. Since there are 3 collisions, the minimum number of time before A’s transmission is 3 slot times.
d) What is the maximum amount of time (measured in slot times) that might elapse from the end of C’s transmission to the beginning of A’s successful transmission? Explain your answer.

If we assume that in each round where a collision occurs, both stations choose the longest possible delay, then the first round will consume one slot, the second round two slots and the third round four slots for a total of 7. In the fourth round, A must delay for less time than B. The longest B can delay is 7 slots, so the longest A can delay is 6 slots. This delay gives a total of 13 slots as the maximum.

e) What is the probability that the complete sequence of collisions and ultimately successful transmission events would occur and complete in the minimal time you described in your answer to part (c)? In this case, a formula like $3/4 \times 7/8 \times (1 - 1/4)$, together with a brief explanation of the formula will probably be of more value than a single number like 3/128.

After C finishes, A and B both transmit with no delay with probability 1.

In the next round, to minimize delay, A and B must both transmit after 0 delay. But they each can choose either 0 delay or 1 slot delay with probability 1/2. Therefore the probability that they both choose 0 is $1/2 \times 1/2 = 1/4$.

By a similar argument, the probability that they both choose slot 0 out of the 4 options they have in the third round is $1/4 \times 1/4 = 1/16$.

Finally, in round 4, A must transmit in slot 0. This happens with probability 1/8 since A has 8 choices. At the same time, for A to succeed, B must not transmit in slot 0. This outcome has probability $1 - 1/8$. Therefore the probability that A succeeds by transmitting in the 0th slot during the fourth round is $1/8 \times (1 - 1/8)$.

Putting all of this together we get the probability of all of these things happening together is

$1/4 \times 1/16 \times 1/8 \times 7/8$
2) This question explores how you can use what you know about Strings to solve a problem from computational biology.

A strand of bacterial messenger RNA can be encoded as a long string of letters representing the four possible bases: adenine (A), guanine (G), cytosine (C), and uracil (U). For example, one short RNA strand is:

\[ \text{UUACAUGUUAACGGACUAGGACGGAUGACAACUAAGAA} \]

Only certain parts of an RNA strand are responsible for protein synthesis-- the rest of the strand is unused while building proteins. The subsequences that are used for protein synthesis can be identified by the fact that they always begin with the triple of bases “AUG”, known as the **start codon** and end with the triple “UAG”, known as the **stop codon**. (Other stop codons exist, but we will ignore them.)

The following is the example from above, rewritten to delineate the important subsequences from the junk:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UU</td>
<td>A</td>
<td>AUG</td>
<td>UUAACGGAC</td>
<td>UAG</td>
<td>GACGG</td>
<td>AUG</td>
</tr>
<tr>
<td>junk</td>
<td>start</td>
<td>stop</td>
<td>junk</td>
<td>start</td>
<td>stop</td>
<td>junk</td>
</tr>
</tbody>
</table>

Real strands are many thousands of bases, and the sections encoding proteins are much longer.

Suppose we wish to write a method `extractSequences` that extracts the sequences of bases between adjacent start and stop codons in a form that would make it easy to display the results. The method takes a String composed only of the letters A, G, C, and U as a parameter, and it returns a String composed of just those subsequences of the original String that were preceded by a start codon and followed by a stop codon. Each sequence you extract should be separated from the next sequence extracted by a newline character. For example, the code

```java
String bunchOfRNA = "UUACAUGUUAACGGACUAGGACGGAUGACAACUAAGAA";
result.setText(extractSequences(bunchOfRNA));
```

would set the result text area to contain:

\[ \text{UUACGGAC} \\
\text{ACACCA} \]

For simplicity, we will assume that a sequence never ends in the middle of a coding region, so every “AUG” will be matched by a “UAG”.
Here is one, correct implementation of `extractSequences`:

```java
public String extractSequences(String rna) {
    int start = rna.indexOf("AUG");
    String result = "";
    while (start >= 0) {
        int end = rna.indexOf("UAG", start);
        result = result + rna.substring(start + 3, end) + "\n";
        start = rna.indexOf("AUG", end + 3);
    }
    return result;
}
```

On the following pages, you will find four programs that are similar but not identical to the program shown above. Each differs from above by a single line of code. The lines containing the changes are the ones identified by arrows. Some of these programs will function exactly like the program shown above. Others will work differently or not at all.

For each program, indicate whether a) it will work like the original program, b) a syntax error will be detected as soon as you compile the program, c) the program will run, but it will not behave like the original (possibly including causing an error to occur while it runs). If there will be an error, identify the line on which it will occur and explain the nature of the error. If no error occurs, but the program behaves differently from the original, explain what would happen if it were applied to the string `bunchOfRNA` described above.

We have placed a second copy of the correct code at the bottom of page 7 to make it easier to compare it to the modified versions.

a)

```java
public String extractSequences(String rna) {
    // Arrow points to this line
    start = rna.indexOf("AUG");
    String result = "";
    while (start >= 0) {
        int end = rna.indexOf("UAG", start);
        result = result + rna.substring(start + 3, end) + "\n";
        start = rna.indexOf("AUG", end + 3);
    }
    return result;
}
```

BlueJ (or any other Java compiler) will reject this method as a syntax error because the variable “start” is not defined. (Alternatively, if start was defined as an instance variable its use by this method is likely to cause a program error).
b)

```java
public String extractSequences(String rna) {
    int start = rna.indexOf("AUG");
    String result = "";
    while (start >= 0) {
        int end = rna.indexOf("UAG", start);
        result = result + rna.substring(start, end) + "\n";
        start = rna.indexOf("AUG", end + 3);
    }
    return result;
}
```

Since `indexOf` sets `start` to the position of the first character of the substring `AUG`, leaving the “+ 3” out of the invocation of `substring` will make it include this start codon at the beginning of each sequence it extracts:

```
AUGUUAACGGAC
AUGACACCA
```

c)

```java
public String extractSequences(String rna) {
    int start = rna.indexOf("AUG");
    String result = "";
    while (start >= 0) {
        int end = rna.indexOf("UAG", start + 3);
        result = result + rna.substring(start + 3, end) + "\n";
        start = rna.indexOf("AUG", end + 3);
    }
    return result;
}
```

The original version of this line asked `indexOf` to start the search at the first character of the start codon `AUG`. The earliest a match for `UAG` could be found was after this start codon. As a result, adding 3 to the start position will not change the results the code produces (but might save a little time)
d)

    public String extractSequences(String rna) {
        int start = rna.indexOf("AUG");
        String result = "";
        while (start >= 0) {
            int end = rna.indexOf("UAG", start);
            result = result + rna.substring(start + 3, end) + "\n";
            start = rna.indexOf("AUG", end + 3);
        }
        return result;
    }

At the last statement of this loop, we know that start indicates the position at which
the previous use of indexOf found a copy of the start codon AUG. As a result, if we
ask it to find the same start codon starting at this position, it will return the current
value of start so that the statement has no effect. This means that the program will get
stuck in an infinite loop repeatedly processing the first copy of the start codon.

// This is a copy of the correct code shown at the top of page 5
// provided to make it possible to compare it to the alternatives b, c, and d without having to flip between pages

    public String extractSequences(String rna) {
        int start = rna.indexOf("AUG");
        String result = "";
        while (start >= 0) {
            int end = rna.indexOf("UAG", start);
            result = result + rna.substring(start + 3, end) + "\n";
            start = rna.indexOf("AUG", end + 3);
        }
        return result;
    }
Reading genetic sequences is made much easier when spaces are added between each codon. Please write a method `insertSpaces` that takes a String composed only of the letters A, G, C, and U as a parameter and returns a String with a space added between each codon. For example,

```
insertSpaces("UUAACGGAC")
```
should return

"UUA ACG GAC"

You may assume that the length of the input is divisible by three.

```java
public String insertSpaces(String rna) {

    String result = "";

    while ( rna.length() > 3 ) {
        result = result + rna.substring( 0, 3 ) + " ";
        rna = rna.substring( 3 );
    }

    return result + rna;
}
```
3) On the following two pages, you will find the definitions of two classes designed to implement a very simple IM program. A sample of the user interface the program will present is shown below:

The program is designed to provide a truly “instant” form of instant messaging. While the program is running, its user can enter his or her own screen name and password, a message, and the screen name of a person to whom the message should be sent. Then, when the user clicks the “Send” button, the program will log in to AOL, send the message, and immediately log out.

Unfortunately, the program isn’t quite complete. In particular, the declarations of several variables and/or formal parameters are missing. The names whose declarations are missing are listed below.

```plaintext
  toAOL
  unnormalized
  message
  send
```

To help you answer this question, we have highlighted the references to these variable in bold face within the code.

What we want you to do is complete the program by adding the missing declarations. We have placed shaded bubbles at all the points in the program where the missing declarations might go. These bubbles can be found after the existing instance variables and at the locations where you might place local variable declarations at the start of each method. In addition, we left room between the parentheses in method headers for you to insert formal parameter declarations as necessary. You will not need to fill in all of the bubbles we have provided, but you should not need to place declarations anywhere outside these bubbles.

Each name should be declared as locally as possible. That is, if the program would work if a given name was declared either as a local variable or an instance variable, then you should declare it as a local variable.
public class Saim extends GUIManager {

    public Saim() {
        this.createWindow( WINDOW_WIDTH, WINDOW_HEIGHT );

        JPanel curPanel;

        // Create a pane containing fields for screenname and password
        curPanel= new JPanel();
        curPanel.add( new JLabel( "Screen Name:" ) );
        curPanel.add( screenName );
        curPanel.add( new JLabel( "   Password:" ) );
        curPanel.add( password );
        contentPane.add( curPanel );

        // Create a pane for the message area and its label
        curPanel = new JPanel();
        curPanel.add( new JLabel( "Message " ) );
        curPanel.add( message );
        contentPane.add( curPanel );

        // Create a panel containing buddy list and send button
        curPanel = new JPanel();
        curPanel.add( new JLabel( "Buddy Name" ) );
        curPanel.add( buddyName );
        send = new JButton( "Send Message" );
        curPanel.add( send );
        contentPane.add( curPanel );
    }
}
// Code to respond when GUI buttons are pressed
public void buttonClicked( JButton which ) {

    FLAPConnection toAOL;

    // Attempt to create a connection and login
    toAOL = new FLAPConnection( normalize(screenName.getText()) );
    toAOL.out.printPacket(
        new TOCSignonPacket( normalize(screenName.getText()), password.getText()).toString() );
    String response = toAOL.in.nextPacket();

    // Determine if connection was accepted
    if ( response.startsWith( "SIGN_ON" ) ) {
        toAOL.out.printPacket( "toc_init_done" );
        // send the message
        toAOL.out.printPacket(
            new TOCSendIMPacket( normalize(buddyName.getText()), message.getText()).toString() );
    } else {
        message.setText( "Unable to connect: " + response );
    }
    toAOL.close();
}

private String normalize( String unnormalized ) {  
    String normalized = unnormalized.toLowerCase();

    while ( normalized.contains( " " ) ) {
        int blankPos = normalized.indexOf( " ");
        normalized = normalized.substring(0, blankPos) + normalized.substring(blankPos+1);
    }
    return normalized;
}

public class TOCSendIMPacket {
    private String buddy;
    private String message;

    public TOCSendIMPacket( String screenName, String text ) {
        buddy = screenName;
        message = text;
    }

    public String toString( ) {
        return "toc2_send_im " + buddy + " \"" + message + "\"" ;
    }
}
4) Consider the three “messages” shown in the table below. Each of these messages is composed of the letters g, i, n, o, r, and s. The table shows how often each of these letters appears in each message. (To keep the problem simple, we will ignore the spaces that occur in the messages.)

<table>
<thead>
<tr>
<th>LETTERS</th>
<th>g</th>
<th>i</th>
<th>n</th>
<th>o</th>
<th>r</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>snoring gorgons</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>rigging rigors</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>gross gnosis</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Below, you will find four possible Huffman trees labeled i, ii, iii, and iv

i.  

ii.  

iii.  

iv.  

a) Show the binary sequence that would be used to encode the word “snoring” using tree i. (i.e. In the table below, fill in the correct binary code for each letter in the word.)

<table>
<thead>
<tr>
<th>LETTER</th>
<th>S</th>
<th>N</th>
<th>O</th>
<th>R</th>
<th>I</th>
<th>N</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>1</td>
<td>010</td>
<td>011</td>
<td>0001</td>
<td>0000</td>
<td>010</td>
<td>001</td>
</tr>
</tbody>
</table>
b) As you know, at certain points in the construction of a Huffman tree, you can choose different ways to proceed. As a result, a given phrase may have several, distinct Huffman trees. In fact, two of the trees shown are both valid Huffman trees for the same phrase. Identify the phrase and the two trees that could be used to generate Huffman codes for this phrase.

Trees ii and iv are both possible Huffman trees for the message “snoring gorgons”.

c) Suppose that a message composed of the five letters a, b, c, d, and e is encoded using a Huffman code. Assume that the message contains more a’s than b’s, more b’s than c’s, more c’s than d’s, and more d’s than e’s. If the binary code used to represent an e is 0001, what codes must be used for the other four letters?

\[ a = 1, \ b = 01, \ c = 001, \ d = 0000 \]
5) For this problem, we would like you to consider two coding schemes in which the amount of time between transitions rather than the presence or absence of energy flow is used to distinguish 0’s and 1’s.

**Morse Binary.** The first scheme is nearly identical to the use of dots and dashes in Morse Code. Basically, we will use dots to represent 0’s and dashes to represent 1’s. To be more precise, we will represent a 0 by turning the power on for 1 time unit and then off for 1 time unit. We will represent a 1 by turning the power on for 2 time units and then off for 1 time unit. As an example, the message 01101 would be transmitted using a signal of the form:

![Signal](image)

Note, that with this scheme, the duration of a bit is not fixed. It takes longer to send a 1 than to send a 0. We will call this scheme “Morse Binary”.

**Transition Spacing Coding.** The second scheme assumes that every transition represents the beginning of a new bit and simply uses the time between transitions to determine whether the bit being sent is a 0 or a 1. In particular, if only one time unit elapses between two transitions, that signal will be interpreted as a 0, but if two time units elapse between a transition, then that signal will be interpreted as a 1. For example, the signal shown above would represent the sequence 0010100010 if interpreted using this scheme. We will call this scheme “Transition spacing coding”.

a) We have seen that the signal above could be interpreted either as an example of “Morse Binary” or of “Transition spacing coding”. Could it also be interpreted as an example of on-off keying? If so, what sequence of bits would it represent? If not, why not? In answering this question, assume that you know that the signal begins at the leftmost edge of the diagram (i.e. don’t worry about start bits or other framing issues).

10110110010110
b) Could the signal shown above be interpreted as an example of Manchester encoding? If so, what sequence of bits would it represent? If not, why not?

NO. Given that the signal begins at the left edge of the diagram, we know that the next first transition represents the beginning of a bit-time and that the second (downward) transition must therefore be the middle of the bit time and the third the end of the bit-time. If this is true, following the rules of Manchester would require that there be a transition at the middle of every other bit-time. In the diagram, however, the signal remains at high for the duration of the next bit-time.

Even if you were willing to assume the signal actually started at low shortly before what the diagram shows, there is no way to divide the signal shown into equal duration bit-times in such a way that a transition occurs in the middle of each bit.

c) Consider the signal

![Signal Diagram]

Indicate which of the four encoding schemes (on-off keying, Manchester, Morse Binary, and Transition spacing) might generate this signal and what sequence of binary digits would it be used to encode. Briefly justify your answer.

ON-OFF KEYING = 1010110010110
TRANSITION SPACING = 0000110010
MORSE BINARY - Impossible because the 3rd point where the signal is low is too long. All inter-bit lows are the same duration in this scheme.
MANCHESTER - It depends. If you assume the signal starts at the left edge of the diagram, then it is invalid because the third point where the signal becomes high lasts too long (there is a mid-bit transition missing). If, on the other hand, you are willing to assume the first transition is the middle of a 1, then the signal can be interpreted as 1110110.
d) One advantage of using Manchester encoding rather than on-off keying is that it is possible for a receiver to automatically adjust its clock’s frequency to remain in synchronization with the transmitter when Manchester encoding is used. Would this be possible with Morse binary? Briefly explain your answer.

Yes. Because the pauses between bits are of fixed length and occur regularly (between every pair of bits) regardless of the sequence of 0s and 1s being transmitted, the receiver could measure these pauses and use these measurements to remain in synchronization with the sender. (Note, I am assuming here that a good approximation of the transmission rate is agreed on between the sender and receiver before communication is attempted. The only issue is to adapt if the sender or receiver’s clock is slightly inaccurate.)

e) We have seen that framing schemes are essential to enable a receiver to identify the beginning and end of a signal when on-off keying. With Manchester encoding, some form of start bit is essential, but it is actually possible for the receiver to identify the end of a message by simply detecting the lack of transitions in the signal. How would the use of Morse binary impact the framing issue? Would a start bit or some other form of start sequence be required to detect the beginning of a transmission. Would it be necessary to use fixed packet sizes or to include a packet length field to identify the end of a transmission? Justify your answer.

No start bit would be required for Morse binary because the absence of a signal is not used to encode either a 0 or a 1. It is only used to separate 0s for 1s.

In practice, it would be unwise to rely on this unless some minimum message length was required. Otherwise, a short burst of static might be interpreted as a message containing just a single 1 or a single 0.