Midterm Exam

2019-12-11 15:42
08d925d

23 October 2019

Solutions

Practice version

Do not write your name on the exam paper. Instead, a sign-in sheet is coming around. Choose any of the sign-in codes, write your name next to it, and then copy that code to the top front of **each page** of the exam.

You have up to 1 hour, 45 minutes. You may use a calculator, but no text book or notes.

1. For each statement below, fill in the blank with the *best* term from the following list. Some terms might be used more than once; some might not be used at all.

   • algorithm • ASCII • binary • bit • Boolean • byte • compression • CPU
   • hexadecimal • input • lossless • lossy • octal • output • pixel • pseudo-code
   • resolution • tree • two's complement • Unicode

   - In the von Neumann architecture, a device that receives data *from* the CPU is called output
   - two's complement is a format for binary numbers that supports both positive or negative numbers.
   - ASCII is a 7-bit code for representing the characters used in American English.
   - A compression technique is called lossy if decompression cannot reproduce the original data perfectly.
   - A(n) tree is a structure in computer science for representing data at branches and leaves.

2. Convert the following base ten (decimal) numbers into binary, using as many bits as needed.

   - `14` = `1110`
   - `25` = `11001`
   - `86` = `1010110`

3. Convert the following 5-bit **signed two's complement** binary numbers into base ten. **Note:** "signed" means that answers **might be negative.**

   - `01011` = `+11`
   - `10011` = `-13`

(over)

- `01111` = `+15`
- `11000` = `-8`
- `00101` = `+5`

4. Add the following pairs of 4-bit **fixed-size unsigned** binary numbers. Your answers must be in binary, but you should check you work by converting to base ten.

```
1                  1                  11
 1001  =   9        0110  =   6        1100  =  12
 1100  =  12        0101  =   5        0111  =   7
------   ----      ------   ----      ------   ----
 0101       5       1011  =  11        0011  =   3
```

5. Convert the following binary number to octal and hexadecimal:

   `1 1 1 1 0 0 0 1 0 1 1 0 0 0 1 1 1 0 0 1 0 0`

   For octal, make groups of 3 from right to left. Then each group is interpreted using the powers 4, 2, 1.

```
1,1 1 1,0 0 0,1 0 1,1 0 0,0 1 1,1 0 0,1 0 0 = 17054344
1   4 2   1 4 2   1 4 2   1 4 2   1 4 2   1 4 2 1
```

   For hexadecimal, make groups of 4 from right to left. Then each group is interpreted using the powers 8, 4, 2, 1.

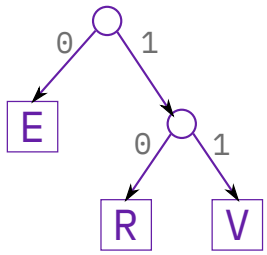
```
1 1,1 1 0 0,0 1 0 1,1 0 0 0,1 1 1 0,0 1 0 0 = 3C58E4
2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1
```

6. Suppose we want to design a custom character encoding just for the word **REVERE**

   (a) How many bits would we need to represent **each distinct letter** if using a **fixed-width** encoding? `2`

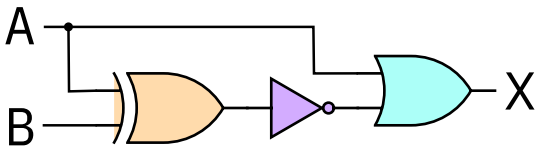
   (b) Using the fixed-width representation in the previous question, how many bits would we need to encode the entire word **REVERE**? `2×6 = 12`

   (c) Draw a tree to represent a **variable-width** encoding of these letters. Use your tree to encode the word **REVERE**. How many bits did you need? ___



100110100 (9 bits)
R EV ER E

7. Which Boolean expression is equivalent to the following circuit diagram?



(a) $X = A' + (A \oplus B)$

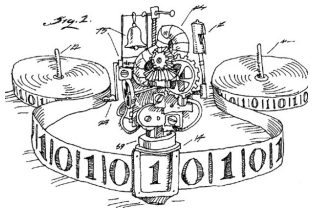
(b) $X = A + (A \oplus B)'$ (This one)

(c) $X = A \oplus (A + B)'$

(d) $X = A + (A \oplus B')$

(over)

8. This problem is about a program for a Turing Machine. Recall that a TM operates by reading and writing symbols on a tape that can be spooled to the left and right. For our program, each cell on the tape can contain either a zero (0), a one (1), or it can be blank (B).



The table below is a representation of a particular TM program. The TM keeps track of its current **state**, a small integer starting at 0.

The first row of the table says that if we're in state 0, and the symbol on the tape at the current position is a 0, we should write a **1** to that position, move the position to the **Right**, and stay in state **0**.

If the "next state" differs from the current state, that represents a **transition**. Use the new state for subsequent operations. Computation continues according to the instructions in the table, until we reach the "halt" state, when the machine stops.

rule number	current state	current symbol	write symbol	move to	next state
1	0	0	1	R	0
2	0	1	0	R	0
3	0	B	B	L	1
4	1	0	1	R	halt
5	1	1	0	L	1
6	1	B	1	L	halt

Simulate the execution of the above Turing Machine program on a tape containing a 4-bit number surrounded by blanks, as shown below. The starting position is underlined (it's the leftmost 1):

⋯	B	B	1	1	0	0	B	B	⋯

What will be the contents of the tape when the machine halts? ___

In tracing the TM, we'll use brackets [] to highlight the current position.

```
B B[1]1 0 0 B B  in state 0 so rule 2: write 0, move R
B B 0[1]0 0 B B  in state 0 so rule 2: write 0, move R
B B 0 0[0]0 B B  in state 0 so rule 1: write 1, move R
B B 0 0 1[0]B B  in state 0 so rule 1: write 1, move R
B B 0 0 1 1[B]B  in state 0 so rule 3: move L, go to state 1
B B 0 0 1[1]B B  in state 1 so rule 5: write 0, move L
```

```
B B 0 0[1]0 B B  in state 1 so rule 5: write 0, move L
B B 0[0]0 0 B B  in state 1 so rule 4: write 1, move R, then halt
B B 0 1[0]0 B B
```

This TM program **negates** a two's-complement number. Using 4-bit two's complement, `1100` is -4, so the machine turned that into `0100` which is $+4$.

(over)

9. Trace the following algorithm, which begins with two integer inputs, represented by variables A and B. Remember to indicate clearly what is *output* versus what is scratch work (memory).

   Initialize the algorithm with $A = 5$ and $B = 9$.

```
1. Let A,B be two positive integers such that A<B
2. Set N to 0
3. Set K to A (copy the value of A into variable K)
4. If K > B then output N and stop.
5. Set N to N + K
6. Set K to K + 1
7. Go back to step 4
```

```
SCRATCH WORK
    Step  1 2 3 4 5 6 7 4 5 6 7 4 5 6 7 4 5 6 7 4 5 6 7 4
       N    0     5      11      18      26      35
       K      5     6       7       8       9      10
       A  5
       B  9


OUTPUT
    35
```

10. Briefly explain the main differences between **linear search** and **binary search**. Which algorithm has the best performance? What must be known about the data to apply each algorithm?

    Linear search looks through a list one item at a time. It can work on any list, but is slow (it takes time proportional to the length of the list). Binary search requires that the list must be in order by whatever we're using as the search criterion. With binary search, we start from the middle item, and then do a comparison to determine whether to search in the front half or the back half. It's much faster, only $log_2 N$ steps.