CISE Qualifying Exam I
8 January 2018

STUDENT EXAM NUMBER:

General Instructions:

1. This is a closed book exam. You have 6 hours to complete the examination. The proctor will tell you when to begin and when to end.

2. To pass the exam, you must successfully complete at least four (4) of the six (6) areas. You may attempt more than 4 areas, but you do not need to do so.

3. Print your student exam number at the top right corner of every answer page submitted. No name or SU ID number should appear on any of the pages you submit.

4. Start each area on a separate page. If an area involves multiple parts, you may answer multiple parts on the same page. Unless specified otherwise, you should answer *all* parts of a given area.

5. When you have finished your exam, please indicate below the areas that you are submitting answers, along with the number of pages you're submitting for each answer.

X Answered the following questions	Number of pages	
		Algorithms
		Architecture/Computer Organization
		Data Structures & Programming
		Digital Logic Circuit Design
		Operating Systems
		Programming Languages

Algorithms

Problem 1 (33%) You have N distinct baskets, where each basket i can hold at most w_i pounds of weight. You have k identical copies of some item, each weighing 1 pound. Design a dynamic programming algorithm to determine how many ways there are to distribute these items into the baskets. (Because the items are identical, it does not matter which item goes into which basket, but only the total number of items in each basket.)

Problem 2 (33%) Suppose G is a finite directed graph. Show that:

G has a topological sort (linearization) *if and only if* G is acyclic.

Problem 3 (34%) Suppose you are trying to measure some scientific phenomena, and you can use a variety of scientific instruments. Each instrument s_i produces radio signals varying in the frequency range $[l_i, h_i]$. If two scientific instruments produce overlapping frequencies, the results that you are trying to measure will be invalid, so these two instruments cannot be operated at the same time. (For example, if one instrument has frequency range $[2,5]$ and the other has frequency range $[1,3]$, then they cannot be operated at the same time.) Design an algorithm to determine the maximum number of instruments that you can operate at the same time, while ensuring that none of the selected instruments have overlapping frequency ranges.

Architecture/Computer Organization

1. Consider a processor with an 8KB data cache which is 2-way set-associative. Each cache line holds 32B data. The memory address has 32 bits with bit 0 representing the MSB and bit 31 representing the LSB.

   (a) How many tag bits are needed for this data cache? Assume the tag field is located at the MSB of the address.

   (b) Consider the following piece of C code:

```
for (i = 0; i < 4; i++)
  {
    x = data3[i];
    for (j = 0; j < 4; j++)
      {
        /* multiply data1 rows by data2 columns */
        data3[ i ] += data1[ i ][ j ] * data2[ j ][ i ];
      }
    data3[i] = x;
  }
```

   Above, data1 and data2 are both 4×4 integer arrays; data3 is a 1×4 vector. The starting addresses of data1, data2, and data3 are 0x14F40, 0x14FA0, and 0x15000, respectively. Assume this is the first time these data are accessed. Variables i, j, and x are mapped to registers during compilation (i.e., there is no memory access when read and write those variables).

   What is the cache hit rate for executing this piece of code?

2. Consider a pipelined processor with 5 stages: IF (Instruction Fetch), ID (Instruction decode/register file read), EX (Execute/address calculation), MEM (Memory access), and WB (register file write back). The register file has 2 read ports and 1 write port. Assume that the instruction set consists of only 4 types of instructions: ADD (add the contents of two registers and store the result in the third one), BEQ (branch if the content of two registers are equal), LW (load the memory contents into a register), and SW (store the content of a register into a memory location). Moreover, the profiled probabilities of their occurrences are: ADD (50%), BEQ (20%), LW (25%), SW (5%).

   (a) Assume that there are no stalls and that all instructions in a program are independent random variables. How often (percentage of cycles) do we need to use all three register ports in the same cycle?

   (b) Consider the following assembly code.

```
ADD   R5, R2, R1    //$R5=$R2+$R1
LW    R4, 4(R5)     //$R4=MEMORY[4+$R5]
LW    R2, 0(R2)     //$R2=MEMORY[$R2]
ADD   R3, R5, R3    //$R3=$R5+$R3
SW    R3, 0(R5)     //MEMORY[$R5]=$R3
```

The initial values in R1, R2, R3, R4 and R5 are 0x10, 0x20, 0x30 0x40 and 0x50, respectively. The memory is initialized so that each byte in the memory stores an unsigned value that equals the lower 8 bits (8 bit LSB) of its address. For example, the byte at address 0x01234576 has the value 0x76. The processor is big Endian.

What will be the content in registers R1–R5 after running the assembly instructions? If there is a memory write, which memory entries are modified, to what values?

(c) The assembly code in part b is written under the assumption that the processor has forwarding and hazard detection. Assume that the processor has no forwarding and hazard detection (but still with the aforementioned 5-stage pipeline architecture).

Will the register and memory content be updated according to the programmer's original intention (i.e. with the assumption that hazard detection and forwarding is supported by the hardware)? Explain why or why not. If your answer is no, please modify the program by inserting NOPs and/or re-arranging instructions so that it generates the expected result. If your answer is yes, can the execution time be reduced by re-arranging the instructions?

Data Structures & Programming

Provide C, C++, Haskell, Java, or ML code to answer the following questions. All the questions have short, straightforward answers. Answers that are too hard to follow shall be marked wrong.

BACKGROUND. We consider binary search trees in which each node has a character key value and also a count of the number of nodes in its *left* subtree. *Example:* See Figure 1 below.

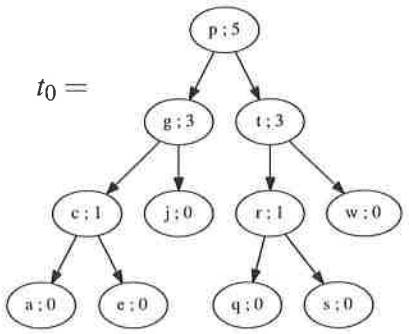

Figure 1.

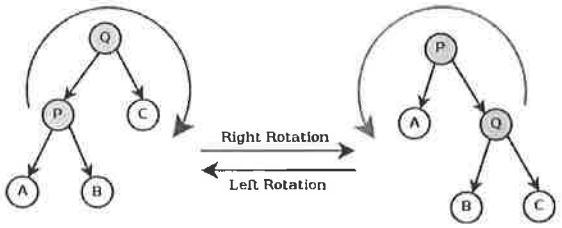

Figure 2

The *depth* of a node N in a binary search tree t is the length of the path from t's root node to N, e.g., in the tree of Figure 1, the `j`-node is of depth 2.

The *index* of a node N in a binary search tree t is the number of nodes to the left of N in t. *Example:* For t_0 = the binary search tree of Figure 1:

node with character	a	c	e	g	j	p	q	r	s	t	w
the t_0-index of that node	0	1	2	3	4	5	6	7	8	9	10

Note: If t_1 is the t_0-subtree rooted at the g-node then, for each node N in t_1, the t_1-index of N = the t_0-index of N; whereas, if t_2 is the t_0-subtree rooted at the `t`-node, then, for each node N in t_2:

$$(\text{the } t_2\text{-index of } N) = (\text{the } t_0\text{-index of } N) - (5+1).$$

CONVENTION. Below, tree $\equiv$ binary search tree.

QUESTION.

(a) ***(5%)*** Give a data-structure for representing (binary search) trees as described above.

(b) ***(21%)*** Write a function **`add`** that takes a tree t and a character c *(which we assume is not in t)* and returns the result of adding c to t. The function should correctly update the left subtree element counts as needed. Your function should run in $O(h)$ time, where h is the height of t.

(c) ***(21%)*** Write a function **`index`** that takes a tree t and a character c *(which we assume occurs in t)* and returns the index of c in t. Your function should run in $O(d)$ time, where d is the depth c's node in t.

(d) ***(21%)*** Write a function **`fetch`** that takes a tree t and a nonnegative integer i *(we assume $i <$ (the number of elements in t))* and returns the character value at index i in tree t. Your function should run in $O(d)$ time, where d is the depth in the tree of the node with index i.

(e) ***(32%)*** Write a function **`reroot`** that takes a tree t and a character c *(which we assume occurs in t)* and returns the result of altering t to make c's node the root. The function should correctly update the left subtree element counts as needed. Your function should run in $O(d)$ time, where d is the depth of c's node in t. ***Hint:*** Tree rotations may be helpful. (See Figure 2 above.)

Digital Logic Circuit Design

Instructions

For this part of the exam, do your work on the special sheets provided. These sheets include the questions for this part of the exam.

Operating Systems

There are three problems in this section. You will be graded on all three.

1. (Memory Management, 35%)

   (a) **See the accompanying answer sheet for this question.**

   (b) Consider a demand-paged computer system where the degree of multiprogramming is currently fixed at four. The system was recently measured to determine utilization of the CPU and the paging disk. Three alternative results are shown below. For each case, what is happening? Can the degree of multiprogramming be increased to increase the CPU utilization? Is the paging helping?

      i. CPU utilization 13%, disk utilization 97%
      ii. CPU utilization 87%, disk utilization 13%
      iii. CPU utilization 13%, disk utilization 3%

   (c) What is an inverted page table and why are they used in modern systems?

2. (CPU Scheduling, 30%)

   (a) Assuming that no process has an infinite loop, which of the following scheduling algorithms could result in starvation? Explain.

      i. Priority (preemptive)
      ii. Round robin (preemptive)
      iii. Shortest Job First

   (b) Shortest Job First scheduling

      i. Prove that SJF is optimal to minimize average response time.
      ii. Why can't pure SJF be implemented? Recommend a heuristic that would implement approximate SJF.

3. Process Coordination (35%)

   (a) Suppose a process calls `V()` (aka `signal()` or `up()`), and there are multiple processes waiting on the semaphore. Discuss two alternatives on choosing what process is selected to run next, citing the pros and cons of each.

   (b) The Spartan Operating System (SOS) only provides binary semaphores. Show how you can use binary semaphores to provide counting semaphores for SOS.

Programming Languages

The syntax for a tiny programming language with simple support for exceptions appears at the bottom of this page. The language's operational semantics and typing rules appear on the next page.

Give a proof by induction (e.g., structural induction, rule induction, or induction on derivations) of the following claim:

Claim: If $\Gamma \vdash E : \mathsf{bool}$, then one of the following four conditions holds:

(1) E is the expression **true**

(2) E is the expression **false**

(3) There is some $\gamma \in \Gamma$ such that E is the expression **raise** γ

(4) There is some expression E' such that $E \longrightarrow E'$

IMPORTANT NOTE: Be sure to **account for all cases** and **include appropriate explanations** in your proof. Do not leave it to the reader to try to determine your intent or to fill in your reasoning.

Syntax The syntax relies on the following two sets:

- The set of ***exceptions***, ranged over by meta-variable γ
- The set of ***expressions***, ranged over by meta-variable E

The specific concrete syntax for exceptions is unimportant for the purposes of this question and is omitted. The abstract syntax for the set of expressions is defined by the following grammar:

$$E \ ::= \ \textbf{true} \mid \textbf{false} \mid \textbf{if}\ E_1\ \textbf{then}\ E_2\ \textbf{else}\ E_3 \mid \textbf{raise}\ \gamma \mid E_1\ \textbf{handle}\ \gamma \Rightarrow E_2$$

Operational Semantics

The operational semantics is given by the transition relation $\longrightarrow$, defined as follows:

$$\textbf{OS1}\ \frac{}{\textbf{if true then } E_2 \textbf{ else } E_3 \longrightarrow E_2} \qquad \textbf{OS2}\ \frac{}{\textbf{if false then } E_2 \textbf{ else } E_3 \longrightarrow E_3}$$

$$\textbf{OS3}\ \frac{E_1 \longrightarrow E_1'}{\textbf{if } E1 \textbf{ then } E_2 \textbf{ else } E_3 \longrightarrow \textbf{if } E1' \textbf{ then } E_2 \textbf{ else } E_3}$$

$$\textbf{OS4}\ \frac{}{\textbf{if (raise } \gamma\textbf{) then } E_2 \textbf{ else } E_3 \longrightarrow \textbf{raise } \gamma}$$

$$\textbf{OS5}\ \frac{}{\textbf{true handle } \gamma \Rightarrow E_2 \longrightarrow \textbf{true}} \qquad \textbf{OS6}\ \frac{}{\textbf{false handle } \gamma \Rightarrow E_2 \longrightarrow \textbf{false}}$$

$$\textbf{OS7}\ \frac{}{\textbf{(raise } \gamma\textbf{) handle } \gamma \Rightarrow E_2 \longrightarrow E_2}$$

$$\textbf{OS8}\ \frac{E_1 \longrightarrow E_1'}{E_1 \textbf{ handle } \gamma \Rightarrow E_2 \longrightarrow E_1' \textbf{ handle } \gamma \Rightarrow E_2}$$

$$\textbf{OS9}\ \frac{}{\textbf{(raise } \gamma\textbf{) handle } \gamma' \Rightarrow E_2 \longrightarrow \textbf{raise } \gamma}\ \gamma \neq \gamma'$$

Typing Rules

A *type context* Γ is a set of exceptions. The typing rules are as follows:

$$\textbf{T1}\ \frac{}{\Gamma \vdash \textbf{true} : \mathsf{bool}} \qquad \textbf{T2}\ \frac{}{\Gamma \vdash \textbf{false} : \mathsf{bool}} \qquad \textbf{T3}\ \frac{}{\Gamma \vdash \textbf{raise } \gamma : \mathsf{bool}}\ \gamma \in \Gamma$$

$$\textbf{T4}\ \frac{\Gamma \vdash E_1 : \mathsf{bool} \qquad \Gamma \vdash E_2 : \mathsf{bool} \qquad \Gamma \vdash E_3 : \mathsf{bool}}{\Gamma \vdash \textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3 : \mathsf{bool}}$$

$$\textbf{T5}\ \frac{\Gamma \cup \{\gamma\} \vdash E_1 : \mathsf{bool} \qquad \Gamma \vdash E_2 : \mathsf{bool}}{\Gamma \vdash E_1 \textbf{ handle } \gamma \Rightarrow E_2 : \mathsf{bool}}\ \gamma \notin \Gamma$$