CMIMC 2016

Computer Science Round

INSTRUCTIONS

1. Do not look at the test before the proctor starts the round.
2. This test consists of 10 short-answer problems to be solved in 60 minutes. Each question is worth one point.
3. Write your name, team name, and team ID on your answer sheet. Circle the subject of the test you are currently taking.
4. Write your answers in the corresponding boxes on the answer sheets.
5. No computational aids other than pencil/pen are permitted.
6. All answers are integers.
7. If you believe that the test contains an error, submit your protest in writing to Porter 100.

CMIMC 2016 Computer Science Reference Sheet

Pseudocode

Code

Code is written with line numbers indicated to the left.
Bold and caps lock indicates a special tag, including but not limited to **TRUE**, **FALSE**, **IF**, **ELSE IF**, **FOR**, **IN**, **WHILE**, **FUNCTION**, **RETURN**, etc.

Variable Assignment

[variable name] ← [value]
This assigns [value] to [variable name].

Conditionals

IF [condition]
  [code]
This executes [code] if [boolean condition] is true.

ELSE IF [condition]
  [code]
This executes [code] if the previous boolean conditions are false and [boolean condition] is true.

ELSE [condition]
  [code]
This executes [code] if all other boolean conditions are false and [boolean condition] is true.

Loops

FOR [variable] **IN** [list]
  [code]
This executes [code] iteratively for each value of [variable] in [list].

WHILE [conditional]
  [code]
This executes [code] iteratively while [conditional] is true.

Functions

FUNCTION [name]([arguments])
This defines a function that takes in [arguments].

RETURN [value]
This causes the function to immediately return [value].

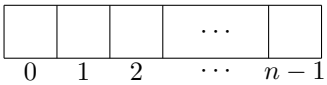
Big O, Ω, Θ

A function $f(n) = O(g(n))$ if there exists a constant $c > 0$ such that $f(n) < c \cdot g(n)$ holds for all sufficiently large n.
A function $f(n) = \Omega(g(n))$ if there exists a constant $c > 0$ such that $f(n) > c \cdot g(n)$ holds for all sufficiently large n.
A function $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Data Structures

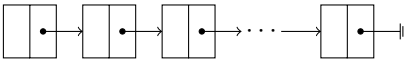
Arrays

A collection of elements accessed by indices.



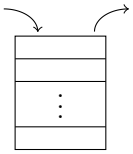
Lists

A sequence of elements, each pointing to the next element and the last one pointing to null.



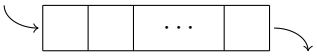
Stacks

A set with a top and bottom where elements are added to the top and removed from the top (first-in last-out).



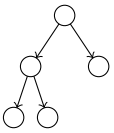
Queues

A set with a front and back where elements are added to the back and removed from the front (first-in last-out).



Trees

A set of elements, each with at most two children.



Graph Theory

A *graph* is a set of vertices V and a set of edges $E \subseteq V \times V$. Two vertices V_1, V_2 are *adjacent* (denoted $V_1 \sim V_2$) if there is an edge connecting them.
The *neighbors* of a vertex is the set of all vertices adjacent to it. The *degree* of a vertex is the number of neighbors.
A *path* is a sequence of vertices $v_0, \ldots, v_k$ such that $v_i \sim v_{i+1}$ for all i and no v_i is visited twice. A *cycle* is like a path but with $v_0 \sim v_k$.
A graph is *connected* if for every two vertices there exists a path from one to the other. Otherwise, it is *disconnected*. The *distance* between any two vertices in a connected graph is the length of the shortest path from one to the other.

Logic

A *boolean function* is a function $f : \{0,1\}^n \mapsto \{0,1\}$. We can write its *truth table* by listing all possible inputs with its outputs.
The three main boolean operators are **NOT** ($\neg$), **AND** ($\wedge$), **OR** ($\vee$), and **XOR** ($\oplus$). Their truth tables are below:

x	y	$\neg x$	$x \wedge y$	$x \vee y$	$x \oplus y$
0	0	1	0	0	0
0	1		0	1	1
1	0	0	0	1	1
1	1		1	1	0

Strings

An *alphabet* is any finite set of symbols. We usually denote an alphabet by Σ.
A *string* is a sequence of elements from an alphabet. We denote the empty string by ε.
The *length* of a string is the length of the sequence it represents. Length is denoted by $|\cdot|$.
For a string $s = s_0 s_1 s_2 \ldots$, the substring starting at i ending at j, denoted $s[i, j)$, is the string $s_i s_{i+1} \ldots s_{j-1}$. This is only well-defined when $0 \le i \le j - 1$, but we also define the substring to be empty if $i = j$.
The concatenation of two strings A and B is the string AB. The concatenation of n copies of the string s is denoted by s^n, where $s^0 = \varepsilon$.

CMIMC 2016

Computer Science

1. For how many distinct ordered triples (a, b, c) of boolean variables does the expression $a \vee (b \wedge c)$ evaluate to true?

2. In concurrent computing, two processes may have their steps interwoven in an unknown order, as long as the steps of each process occur in order. Consider the following two processes:

Process	A	B
Step 1	$x \leftarrow x - 4$	$x \leftarrow x - 5$
Step 2	$x \leftarrow x \cdot 3$	$x \leftarrow x \cdot 4$
Step 3	$x \leftarrow x - 4$	$x \leftarrow x - 5$
Step 4	$x \leftarrow x \cdot 3$	$x \leftarrow x \cdot 4$

   One such interweaving is $A1$, $B1$, $A2$, $B2$, $A3$, $B3$, $B4$, $A4$, but $A1$, $A3$, $A2$, $A4$, $B1$, $B2$, $B3$, $B4$ is not since the steps of A do not occur in order. We run A and B concurrently with x initially valued at 6. Find the minimal possible value of x among all interweavings.

3. Sophia writes an algorithm to solve the graph isomorphism problem. Given a graph $G = (V, E)$, her algorithm iterates through all permutations of the set $\{v_1, \ldots, v_{|V|}\}$, each time examining all ordered pairs $(v_i, v_j) \in V \times V$ to see if an edge exists. When $|V| = 8$, her algorithm makes N such examinations. What is the largest power of two that divides N?

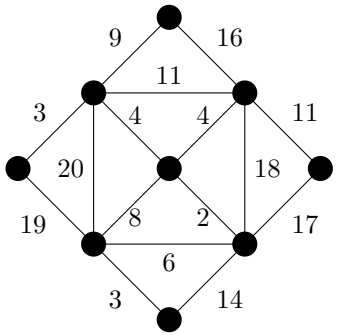
4. Given a list A, let $f(A) = [A[0] + A[1], A[0] - A[1]]$. Alef makes two programs to compute $f(f(...(f(A))))$, where the function is composed n times:

```
1: FUNCTION T_1(A, n)
2:     IF n = 0
3:         RETURN A
4:     ELSE
5:         RETURN [T_1(A, n-1)[0] + T_1(A, n-1)[1],
           T_1(A, n-1)[0] - T_1(A, n-1)[1]]
```

```
1: FUNCTION T_2(A, n)
2:     IF n = 0
3:         RETURN A
4:     ELSE
5:         B ← T_2(A, n-1)
6:         RETURN [B[0] + B[1], B[0] - B[1]]
```

   Each time T_1 or T_2 is called, Alef has to pay one dollar. How much money does he save by calling $T_2([13, 37], 4)$ instead of $T_1([13, 37], 4)$?

5. We define the *weight* of a path to be the sum of the numbers written on each edge of the path. Find the minimum weight among all paths in the graph below that visit each vertex precisely once:



6. Aaron is trying to write a program to compute the terms of the sequence defined recursively by $a_0 = 0$, $a_1 = 1$, and

$$a_n = \begin{cases} a_{n-1} - a_{n-2} & n \equiv 0 \pmod 2 \\ 2a_{n-1} - a_{n-2} & \text{else} \end{cases}$$

However, Aaron makes a typo, accidentally computing the recurrence by

$$a_n = \begin{cases} a_{n-1} - a_{n-2} & n \equiv 0 \pmod 3 \\ 2a_{n-1} - a_{n-2} & \text{else} \end{cases}$$

For how many $0 \le k \le 2016$ did Aaron coincidentally compute the correct value of a_k?

7. Given the list

$$A = [9, 12, 1, 20, 17, 4, 10, 7, 15, 8, 13, 14],$$

we would like to sort it in increasing order. To accomplish this, we will perform the following operation repeatedly: remove an element, then insert it at any position in the list, shifting elements if necessary. What is the minimum number of applications of this operation necessary to sort A?

8. Consider the sequence of sets defined by $S_0 = \{0, 1\}, S_1 = \{0, 1, 2\}$, and for $n \ge 2$,

$$S_n = S_{n-1} \cup \{2^n + x \mid x \in S_{n-2}\}.$$

For example, $S_2 = \{0, 1, 2\} \cup \{2^2 + 0, 2^2 + 1\} = \{0, 1, 2, 4, 5\}$. Find the 200th smallest element of S_{2016}.

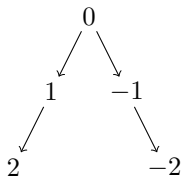
9. Ryan has three distinct eggs, one of which is made of rubber and thus cannot break; unfortunately, he doesn't know which egg is the rubber one. Further, in some 100-story building there exists a floor such that all normal eggs dropped from below that floor will not break, while those dropped from at or above that floor will break and cannot be dropped again. What is the minimum number of times Ryan must drop an egg to determine the floor satisfying this property?

10. Given $x_0 \in \mathbb{R}$, $f, g : \mathbb{R} \to \mathbb{R}$, we define the *non-redundant binary tree* $T(x_0, f, g)$ in the following way:

    (a) The tree T initially consists of just x_0 at height 0.

    (b) Let $v_0, \ldots, v_k$ be the vertices at height h. Then the vertices of height $h + 1$ are added to T by: for $i = 0, 1, \ldots, k$, $f(v_i)$ is added as a child of v_i if $f(v_i) \notin T$, and $g(v_i)$ is added as a child of v_i if $g(v_i) \notin T$.

For example, if $f(x) = x + 1$ and $g(x) = x - 1$, then the first three layers of $T(0, f, g)$ look like:



If $f(x) = 1024x - 2047\lfloor x/2 \rfloor$ and $g(x) = 2x - 3\lfloor x/2 \rfloor + 2\lfloor x/4 \rfloor$, then how many vertices are in $T(2016, f, g)$?