Iverson computing competition
2015 may 26

name _____

school _____
city _____
grade _____
cs teacher _____

are you taking AP or IB computer science? (yes/no) _____
have you taken advanced level courses? (3000 level, e.g. CSE3110 iterative algorithms I.) (yes/no/currently taking) _____

illegible answers will not be marked

question	- - - -	marks	your score
1	continents	10	
2	permutations	10	
3	pebble game	10	
4	fractions	3	
total	- - - -	33	

Exam Format

This is a two-hour paper and pencil exam. There are four questions, each with at least one part. Some part(s) might be easy. Solve as many parts of as many questions as you can.

Programming Language

Questions that require programming can be answered using any language (e.g. C/C++, Java, Python, ...) or pseudo-code. **Pseudo-code should be detailed enough to allow for a near direct translation into a programming language.** Clarify your code with appropriate comments. For full marks, a question must be correct, well-explained, and as simple as possible.

Our primary interest is in thinking skill rather than coding wizardry, so logical thinking and systematic problem solving count for more than programming language knowledge.

Suggestions

1. You can assume that the user enters only valid input in the coding questions.

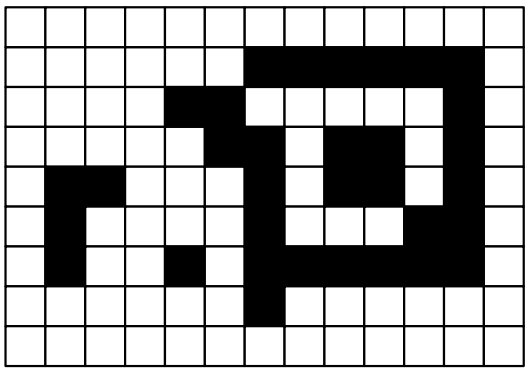
2. In somes cases, sample executions of the desired program are shown. Review the samples carefully to make sure you understand the specifications. The samples may give hints.

3. Design your algorithm before writing any code. Use any format (pseudo-code, diagrams, tables) or aid to assist your design plan. We may give part marks for legible rough work, especially if your final answer is lacking. We are looking for key computing ideas, not specific coding details, so you can invent your own "built-in" functions for subtasks such as reading the next number, or the next character in a string, or loading an array. Make sure to specify such functions by giving a relationship between their inputs and outputs.

4. Read all questions before deciding which ones to attempt, and in which order. Start with the easiest parts of each question.

question 1: continents

Pixel-world is a peculiar planet. It is a flat world that is divided into a grid of square cells. Each cells is completely covered by land or completely covered by water. Example: in the grid below, the dark cells represent land and the light cells represent water.

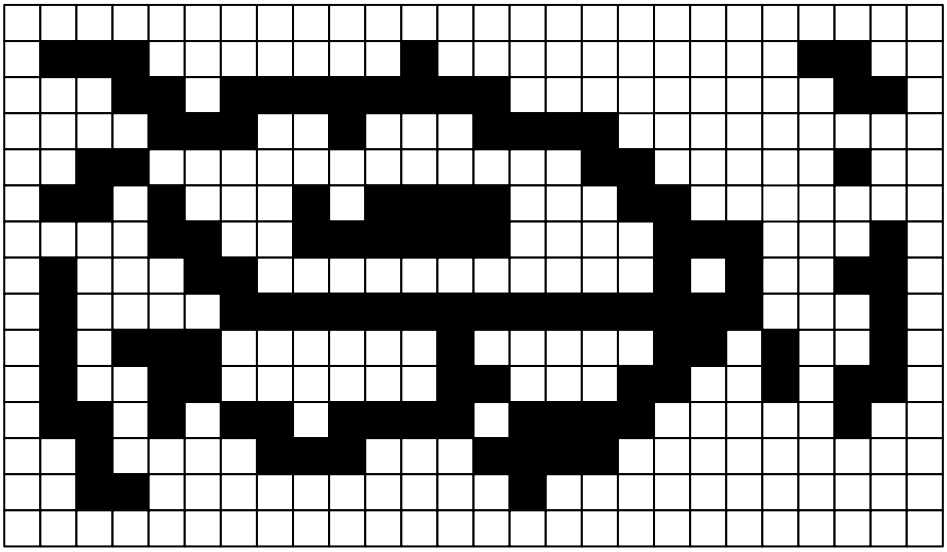


A continent is a collection of land cells, say C, such that

- it is possible to walk between any two cells in C by taking horizontal or vertical steps (no diagonal steps) and never entering a water cell.
- every land cell that can be reached in this way from a land cell in C is also in C.

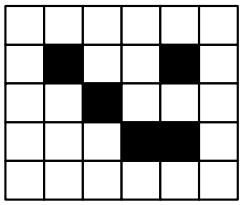
The size of the continent C is the number of cells in C. Example: the grid shown above has 4 continents, with sizes 1, 4, 4, 24.

a) [2 marks] Give the number of continents in the following grid.



b) [2 marks] Suppose, for each land cell, we already know the size of the continent that includes that cell. These sizes are stored in an array and the array is sorted.

Example: `1, 1, 1, 2, 2` is the sorted list of these sizes for the following grid. The size `2` appears twice because it is entered into the array once for each land cell in the continent.



The list below was obtained in this way from a grid with 30 land cells. How many continents does this grid have? Explain briefly.

```
1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4
```

c) [3 marks] Write a function `count_continents(sizes, n)` where `sizes` is a sorted array of `n` integers, obtained from a grid with `n` land cells. The function should return the number of continents in the grid.

d) [3 marks] We represent a grid by specifying two positive integers, `rows` and `columns` — indicating the grid size — together with a two-dimensional array `grid`. For `1` $\leq$ `r` $\leq$ `rows` and `1` $\leq$ `c` $\leq$ `columns`, `grid[r][c]` is `0` if the cell at location `(r, c)` is water, and is `1` if that cell is land.

Write a function `continent_size(r, c, rows, columns, grid)` that returns the size of the continent that includes the cell at location `(r, c)`. If this is a water cell, return `0`. You may assume that every cell on the boundary of the grid is water.

question 2: divisibility testing

In this question, we describe numbers in both decimal and binary form. A decimal number is subscripted with 10; a binary number is subscripted with 2. Example: $5_{10} = 101_2$.

You might know some divisibility tests for decimal numbers. Example: a number is divisible by 3 if and only if the sum of its decimal digits is divisible by 3. This rule does not hold for binary digits: $3_{10} = 11_2$ but the sum of its binary digits is 2_{10}.

We will use *finite state machines* (FSMs) for our tests. A FSM is a computational device that reads in a string of bits (`0` or `1`) and decides whether to accept that string. It has these features:

- **states**, which are depicted as labelled circles (`a, b, c` in the example below);
- for each state `x`, there are precisely two **arcs** that start at `x` and point to some other state (perhaps `x` again); one arc is labelled `0` and the other `1`;
- one state is the **start** state and one is the **accepting** state; the start state has an arrow pointing to it labelled `start`; the accepting state has a thick border; the start and accepting states can be the same.

A computation with a FSM is simple to describe. A "current state" v is maintained which is initialized to be the start state. The input string is read one bit at a time, from left to right. When a bit `b` is read, the current state v is updated to be the state that is pointed to from v by the arc labelled `b`.

Once the entire input is processed, the FSM *accepts* the string if v is the accepting state, otherwise it *rejects* the string. We assume that the input string has at least one bit.

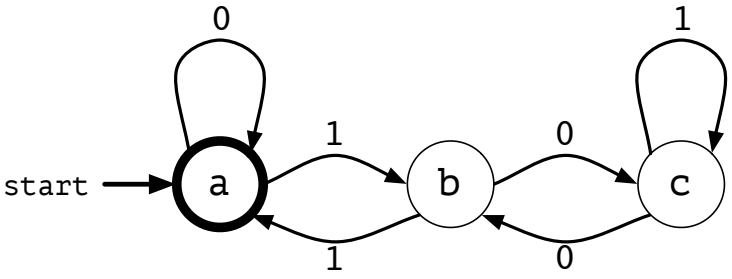


Figure 1: A finite state machine that accepts binary numbers divisible by 3. Here, `a` is both the start state and the accepting state.

Given a FSM, we can illustrate a computation by writing the sequence of states assumed by v, connecting consecutive labels with an arrow that labelled by the associated input bit. Example: for the FSM above, here is the computation for input string `110`:

$$\texttt{a} \xrightarrow{\texttt{1}} \texttt{b} \xrightarrow{\texttt{1}} \texttt{a} \xrightarrow{\texttt{0}} \texttt{a}$$

and here is the computation for input string `100`:

$$\texttt{a} \xrightarrow{\texttt{1}} \texttt{b} \xrightarrow{\texttt{0}} \texttt{c} \xrightarrow{\texttt{0}} \texttt{b}$$

String `110` is accepted because the final state is the accepting state, but string `100` is rejected. In fact, this FSM accepts precisely the binary numbers that are divisible by $3_{10} = 11_2$. Notice that both `011` and `00` are accepted; leading 0s are allowed.

a) [2 marks] Illustrate the computation of the above FSM for (i) input string `1010010` and (ii) input string `1011010`. For each string, state whether it is accepted.

b) [2 marks] Draw a FSM that accepts precisely the binary numbers that are divisible by 2_{10} (leading zeros are allowed).

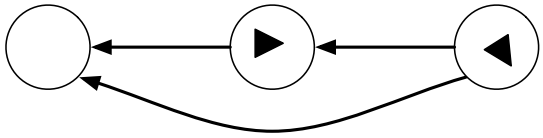
c) [3 marks] Draw a FSM that accepts precisely the binary numbers that are divisible by 5_{10} (leading zeros are allowed).

d) [3 marks] Draw a FSM from the description that accepts precisely binary numbers that are divisible by 3_{10} and either are the number zero or have a leading 1. Example: `11` and `0` are accepted; `011`, `00`, `10` are rejected.

question 3: pebble game

Alice and Bob play a 2-player game. A number of pebbles are placed in various *positions* that are arranged horizontally. On a *turn*, a player moves one pebble and to the left. However, not all such moves are valid; valid moves are specified as part of the game. If no pebble can be moved, then the player whose turn it is loses and the other player wins.

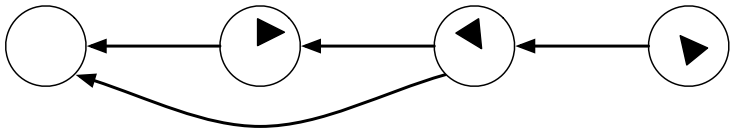
Example: here is a game with 3 positions (circles) and 2 pebbles (triangles). The arcs show the valid moves.



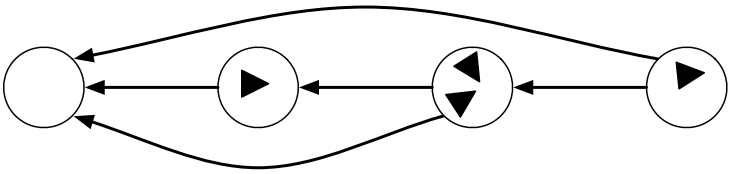
Here, Alice can win the game by moving the rightmost pebble to the middle. Now Bob's only option is to move one of these pebbles to the leftmost point; then Alice moves the other pebble left, and Bob has no moves so Alice wins.

In the following questions, assume both Alice and Bob play perfectly. That is, if the current player can move so that that they can win by continuing to play perfectly, then they make a winning move.

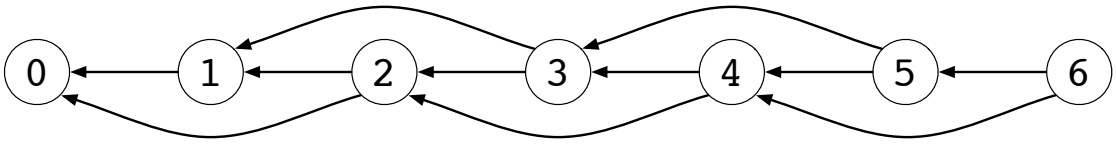
a) [2 marks] Who wins this game? Remember, Alice plays first. Explain briefly. It might help to label the pebbles.



b) [2 marks] Who wins this game? Explain briefly.



c) [3 marks] Consider the game with n positions numbered from 0 to $n-1$ (left to right) and only these valid moves: for $0 \le v \le n-1$, a pebble can move from v to $v-1$; for $v \ge 2$, a pebble can move from v to $v-2$. The game with $n = 7$ is shown below.



Write a function `winner(i, j)` that determines who wins on such a board with one pebble at location $i \ge 0$ and one pebble at location $j \ge 0$, where possibly $i = j$. Return a string, either `"Alice"` or `"Bob"`.

Explain briefly why your code is correct. For full marks, your algorithm should run in a fraction of a second, even for i and j as large as 10^9. *Hint*: there is a nice pattern.

d) [3 marks] Now consider the same board as the previous part, except we may have many pebbles on the board. Write pseudocode for a function `win(a, m, n)` where `n` is the number of positions on the board and `a[]` is an array with `m` nonnegative integers, each between 0 and `n-1`, specifying the initial placement of the pebbles. Again, this code should run quickly and you must explain briefly why it is correct.

question 4: fractions

You are given a fraction $\frac{a}{b} \geq 0$. However, a and b might be large. So, given a positive integer N, you want to find a "simpler" fraction $\frac{c}{d}$ that is as close to $\frac{a}{b}$ as possible but with $0 \leq c, d \leq N$. That is, you should find c and d so that $0 \leq c \leq N$, $0 < d \leq N$, and $\left|\frac{a}{b} - \frac{c}{d}\right|$ as small as possible. If there are multiple solutions, choose the answer such that $c + d$ is as small as possible.

[3 marks] Write a function `simpler(a,b,N)` that prints `closest simpler fraction c / d`

where `c` and `d` are the numerator and denominator of the answer. Remember, you can write helper functions.

Example: calling `simpler(5, 7, 5)` prints `closest simpler fraction 3 / 4`

and calling `simpler(11, 17, 7)` prints `closest simpler fraction 2 / 3`