COSC 2006 FA 2020 SOLUTION
Midterm Exam
Distributed: October 26, 2020
Due: October 28, 2020, at the end of the day

100 marks

INSTRUCTIONS -- Please read carefully before beginning the exam.

- Write your answers in a Word file.
- If you wish, you may use the template `LastName_COSC2006_FA2020_MidtermExam.docx`.
- Number each question.
- Ensure that you have your name and student number on the top of your document.
- Name your document as `LastName_COSC2006_FA2020_MidtermExam.doc(x)`. Make sure to change `LastName` to your last name.
- For Q2, use the template `LastName_COSC2006_FA2020_computeF_recursive.cpp`. Before you submit, replace the `LastName` prefix with your last name.
- Submit the exam through the BB submission site under Tests & Surveys → Midterm Exam.
- Use Equation Editor where appropriate, or, at least, use symbols when required. For instance, ***do not*** write a^2 x b^2 = c^2, but use symbols, italics, and superscripts to write $a^2 \times b^2 = c^2$ (Equation Editor was not required in this case).
- You may copy/paste from the template `COSC2006-FA2020-EquationEditor-Symbol-Template.docx` that is available on BB if you need assistance with Equation Editor and Symbols.
- Do not use first- or second-person. Please use passive voice. For instance, do not write "We multiply a by b…" or "You would multiply a by b…". Instead, write "*a* and *b* are multiplied…".
- For questions that require code, name the code as specified in the question and submit the source file(s) to the same BB Midterm Exam submission site.
- Show all work.
- Attempt each question. Partial credit will be awarded.
- Provide your best effort for each question.
- You may use your text, notes, lectures, the practice exercises, and a calculator and/or Python, but you may not use any other resources, including (and especially) Internet resources.
- It is also an individual exam, and therefore you may not consult with your colleagues, other faculty, or anyone else.
- Please avoid posing questions to the instructor during the test. The questions are straightforward and are easily interpreted. If you feel that you may be misinterpreting a question, simply show all your work and provide justification(s) for your answer(s). Partial credit is awarded.
- If for some reason it becomes necessary to clarify a question, a BB announcement/email will be sent to the class during the duration of the exam.

1. **(10 marks)** Prove that $7^n + 2$ is divisible by 3 for all non-negative integers n. Show all work.

SOLUTION

Prove that $7^n + 2 = 3m$ for some positive integer m.
Base case: $n = 1 \Rightarrow 7^n + 2 = 7 + 2 = 9 = 3(3)$ ($m = 3$).
$\therefore$ The base case is true. (***Note***: The base case is also true for $n = 0$).

Induction hypothesis: Assume that $7^k + 2 = 3m$ for $k > 1$ and some positive integer m.

Induction step: For $k + 1$, show that $7^{k+1} + 2 = 3m'$ for some positive integer m', given the I.H.

Also from the I.H., it is known that $7^k = 3m - 2$.

$7^{k+1} + 2 = 7(7^k) + 2 = 7(3m - 2) + 2 = 21m - 14 + 2 = 21m - 12 = 3(7m - 4)$. $\therefore$ $m' = 7m + 4$, which is a positive integer.

$\therefore$ $7^n + 2 = 3m$ for some positive integer m. QED

NOTE: As a demonstration, assume that $n = 4$. Then, $7^4 + 2 = 2403$, so that $m = 801$. Therefore, $m' = 7(801) - 4 = 5603$, so that $7^5 + 2 = 16807 + 2 = 16809 = 3(5603)$.

2. **(14 marks total)** Consider the following recursive definition.

$F(1, 1) = 1$
$F(2, 2) = 1$
$F(2, 1) = 0$
$F(n + 1, 1) = -F(n - 1, 1)$ for $n > 2$
$F(n + 1, n + 1) = 2F(n, n)$ for $n \geq 2$
$F(n + 1, m + 1) = 2F(n, m) - F(n - 1, m + 1)$ for $n > 2$

a. **(10 marks)** Write a C++ program to implement this recursion. *Determine* the number of recursive function calls required for each $F(n, m)$, clearly commenting your code and identifying the base case(s). Use the template `LastName_COSC2006_FA2020_computeF_recursive.cpp` on BB. Replace `LastName` with your last name.

Use your program for the following:

b. **(1 mark)** Calculate the value of $F(10, 4)$ and the number of recursive function calls required.

From the recursive function, $F(10, 4) = -120$, requiring 59 recursive function calls.

c. **(1 mark)** Calculate the value of *F*(14, 2) and the number of recursive function calls required.

From the recursive function, *F*(14, 2) = 13, requiring 34 recursive function calls.

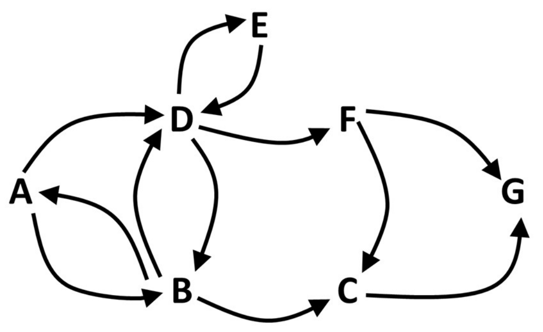
d. **(1 mark)** Calculate the value of *F*(31, 19) and the number of recursive function calls required.

From the recursive function, *F*(31, 19) = 577372160, requiring 556852 recursive function calls.

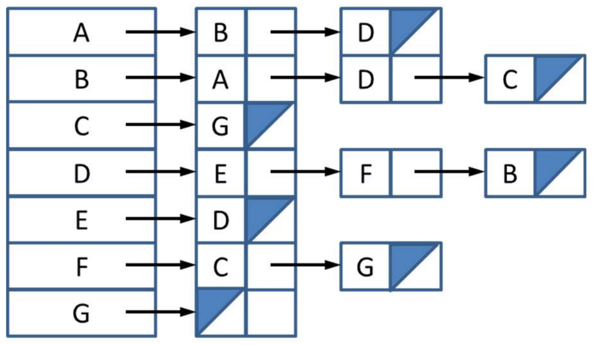
e. **(1 mark)** Calculate the value of *F*(54, 44) and the number of recursive function calls required.

From the recursive function, *F*(54, 44) = 0, requiring 14356760 recursive function calls.

3. **(10 marks)** Determine and draw the adjacency list ***using linked bags*** for the flight map below, using the format and notation described in the lecture Chapter 6 – Stacks (2). You may use text, arrows, etc., to draw the adjacency list.



SOLUTION



4. **(12 marks)** Using the recursive definition of $\binom{n}{k}$, compute $\binom{5}{3}$ through ***memoization***. The algorithm does not need to be implemented in code, but you will work through the steps. Show the final memoization matrix (i.e. the 2D array required for the memoization). Place the values of n in rows, and k in columns. Show all steps, starting with $n = 5$ and $k = 3$.

For simplicity, you may use the notation $C(n, k)$ for $\binom{n}{k}$. For example, you may write $\binom{5}{3}$ as $C(5, 3)$.

Describe each step, calculation, and update to the memoization matrix (2D array).

Hint: The $(n + 1) \times (k + 1)$ 2D array will have the following form:

$$\text{memoize[6][4]} = \overbrace{\left[\begin{array}{c} \\ \\ \\ \end{array}\right]}^{k+1 \text{ columns}} \left.\vphantom{\begin{array}{c} \\ \\ \\ \end{array}}\right\} n + 1 \text{ rows}$$

SOLUTION

Recursive definition: $C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$

$C(5, 3) = C(4, 2) + C(4, 3)$

Start with $C(4, 2)$:
$C(4, 2) = C(3, 1) + C(3, 2)$
$C(3, 1) = C(2, 0) + C(2, 1)$

Base case: $C(n, 0) = 1$, and therefore $C(2, 0) = 1$, so add the calculation for $C(2, 0)$ to the memoization matrix.

$$\text{memoize[6][4]} = \overbrace{\left[\begin{array}{c} \mathbf{1} \\ \\ \\ \end{array}\right]}^{k+1 \text{ columns}} \left.\vphantom{\begin{array}{c} \\ \\ \\ \end{array}}\right\} n + 1 \text{ rows}$$

$C(3, 1) = C(2, 0) + C(2, 1)$

$C(2, 1) = C(1, 0) + C(1, 1)$

Base cases were reached. *C*(*n*, 0) = 1, and therefore *C*(1, 0) = 1. Additionally, *C*(*n*, *n*) = 1, and therefore *C*(1, 1) = 1. These two entries can be added to the memoization matrix:

$$\text{memoize[6][4]} = \left.\overbrace{\begin{bmatrix} \mathbf{1} & \mathbf{1} & & \\ 1 & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}}^{k+1 \text{ columns}}\right\} n+1 \text{ rows}$$

C(2, 1) = *C*(1, 0) + *C*(1, 1) = 1 + 1 = 2, so the entry for *C*(2, 1) can be added to the memoization matrix.

$$\text{memoize[6][4]} = \left.\overbrace{\begin{bmatrix} 1 & 1 & & \\ 1 & \mathbf{2} & & \\ & & & \\ & & & \\ & & & \end{bmatrix}}^{k+1 \text{ columns}}\right\} n+1 \text{ rows}$$

Now, *C*(3, 1) = *C*(2, 0) + *C*(2, 1) = 1 + 2 = 3, so the entry for *C*(3, 1) can be added to the memoization matrix.

$$\text{memoize[6][4]} = \left.\overbrace{\begin{bmatrix} 1 & 1 & & \\ 1 & 2 & & \\ & \mathbf{3} & & \\ & & & \\ & & & \end{bmatrix}}^{k+1 \text{ columns}}\right\} n+1 \text{ rows}$$

Going back to *C*(4, 2) = *C*(3, 1) + *C*(3, 2), it is seen that *C*(3, 2) needs to be calculated.

C(3, 2) = *C*(2, 1) + *C*(2, 2)

The base case of *C*(*n*, *n*) = *C*(2, 2) = 1 is reached, so the entry for *C*(2, 2) is added to the memoization matrix.

$$\text{memoize[6][4]} = \left.\overbrace{\begin{bmatrix} 1 & 1 & & \\ 1 & 2 & \mathbf{1} & \\ & 3 & & \\ & & & \\ & & & \end{bmatrix}}^{k+1 \text{ columns}}\right\} n+1 \text{ rows}$$

$\therefore$ $C(3, 2) = C(2, 1) + C(2, 2) = 2 + 1 = 3$, so the entry for $C(3, 2)$ can be added to the memoization matrix.

$$\text{memoize[6][4]} = \left.\overbrace{\begin{bmatrix} 1 & 1 & & \\ 1 & 2 & 1 & \\ & 3 & \mathbf{3} & \\ & & & \\ & & & \end{bmatrix}}^{k+1 \text{ columns}}\right\} n+1 \text{ rows}$$

Now, $C(4, 2) = C(3, 1) + C(3, 2)$ can be calculated as $3 + 3 = 6$, so the entry for $C(4, 2)$ can be added to the memoization matrix.

$$\text{memoize[6][4]} = \left.\overbrace{\begin{bmatrix} 1 & 1 & & \\ 1 & 2 & 1 & \\ & 3 & 3 & \\ & & \mathbf{6} & \\ & & & \end{bmatrix}}^{k+1 \text{ columns}}\right\} n+1 \text{ rows}$$

$\therefore C(5, 3) = C(4, 2) + C(4, 3) = 6 + C(4, 3)$, so $C(4, 3)$ needs to be calculated.

$C(4, 3) = C(3, 2) + C(3, 3)$. The base case of $C(n, n)$ has been reached for $n = 3$, and therefore $C(3, 3) = 1$. This entry can be added to the memoization matrix.

$$\text{memoize[6][4]} = \left.\overbrace{\begin{bmatrix} 1 & 1 & & \\ 1 & 2 & 1 & \\ & 3 & 3 & \mathbf{1} \\ & & 6 & \\ & & & \end{bmatrix}}^{k+1 \text{ columns}}\right\} n+1 \text{ rows}$$

Consequently $C(4, 3) = 3 + 1 = 4$, and can be added to the memoization matrix.

$$\text{memoize[6][4]} = \left.\overbrace{\begin{bmatrix} & & & \\ 1 & 1 & & \\ 1 & 2 & 1 & \\ & 3 & 3 & 1 \\ & & 6 & \mathbf{4} \\ & & & \end{bmatrix}}^{k+1\text{ columns}}\right\} n+1 \text{ rows}$$

Finally, $C(5, 3) = C(4, 2) + C(4, 3) = 6 + 4 = 10$, and the final entry is added to the memoization matrix.

$$\text{memoize[6][4]} = \left.\overbrace{\begin{bmatrix} & & & \\ 1 & 1 & & \\ 1 & 2 & 1 & \\ & 3 & 3 & 1 \\ & & 6 & 4 \\ & & & \mathbf{10} \end{bmatrix}}^{k+1\text{ columns}}\right\} n+1 \text{ rows}$$

Fill-in-the-Blanks/Converse Definitions. Write the answer in your submission document.

5. **(2 marks)** **COBOL (COmmon Business Oriented Language)** is a high-level programming language originally developed in the late 1950s and that is used primarily for business applications (general knowledge question for computer science).

6. **(2 marks)** In OO analysis **coupling** is a measure of dependence among modules.

7. **(2 marks)** The publicly available methods and data for a class is the **interface** of the class.

8. **(2 marks)** Deleting a pointer that points to a large block of memory allocated from the heap results in a condition known as a/an **memory leak**.

9. **(2 marks)** **8361** recursive function calls are needed to compute *Fib*(19) recursively.

10. **(2 marks)** **37** recursive function calls are needed to compute *Fib*(19) recursively with memoization.

11. **(2 marks)** **18** iterations are needed to compute *Fib*(19) iteratively.

12. **(10 marks)** For three iterations, determine the values of `first`, `last`, and `mid` of the recursive binary search algorithm for the eleven values shown below. Show all work. The target value is -15.

```
-19  -11  0  1  2  6  10  11  13  29 120
```

SOLUTION

```
              Initially        After 1 iteration       After 2 iterations

first:        0                0                       0
last:         10               4                       1
mid:          (0+10)/2 = 5     (0 + 4)/2 = 2           (0 + 1)/2 = 0
```

Assume that `A` is the array,

For iteration 1, `A[mid]` = `A[5]` = 6 > -15.
`first` ← 0
`last` ← `mid` – 1 = 5 – 1 = 4
`mid` ← (0 + 4) / 2 = 2

For iteration 2, `A[mid]` = `A[2]` = 0 > -15.
`first` ← 0
`last` ← `mid` – 1 = 2 – 1 = 1
`mid` ← (0 + 1) / 2 = 0

Initially: A[first] = -19, A[last] = 120, A[mid] = 6
Iteration 1: A[first] = -19, A[last] = 2, A[mid] = 0
Iteration 2: A[first] = -19, A[last] = -11, A[mid] = -19

13. **(5 marks)** Compare and contrast a deep copy and a shallow copy.

SOLUTION

In a shallow copy, a new pointer is allocated that references existing memory. In a shallow copy of an object, usually only the data members are copied. A copy constructor is not required.

In a deep copy, the existing memory of an object is copied into another area of memory so that there are two or more copies of the same memory. A deep copy requires a copy constructor.

14. **(10 marks)** Any recursive algorithm can be reformulated / rewritten as an iterative algorithm. Provide an informal "proof" of this statement. You can do this by showing how recursive algorithms can be "converted" into iterative algorithms. (***Hint***: There are two cases that you need to consider).

SOLUTION

For tail recursion, a straightforward iterative implementation can be formulated. If there is no tail recursion, the function can be converted to iteration using a stack, as demonstrated in the ToH problem, which was solved both iteratively and using a stack. Calling a recursive function places the calling function state into stack memory (a reason that deep recursions may lead to programs ending due to running out stack memory). Consequently, using a stack data structure simulates the behaviour of recursion.

15. **(3 marks)** Suppose that an array of strings is initialized as follows:

```
include <string>
string s[] = {'abc', 'abcde', '10101011', 'COSC2006-FA2018', ..., '3.14159'};
int Ns;      // Length of string
```

Complete the following code, wherein `Ns` is calculated (not assigned).

SOLUTION

```
Ns = sizeof(s)/sizeof(*s);
```

16. **(4 marks)** What is the problem with the following recursive function?

```
int example2 (int n, int m)
{
   if (n == 0)
      return (0);
   else
   {
      m += n;
      return (example2(n - 2, m) + (4*m));
   }
}
```

SOLUTION

If *n* is an odd number, the base case will never be reached because the recursive call `example2(n - 2, m)` will result in another odd number (*n* – 2 is odd if n is odd), and therefore the even number *n* = 0. The problem can be rectified by adding another base case for the *n* = 1 or *n* = -1, or by expanding the conditional in the original base case.

17. **(8 marks)** Consider the array-based stack `Astack` shown below.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value	5	3	17	-1	-9	11							

a. **(3 marks)** What does `std::cout << Astack.peek();` display?

SOLUTION

11 (the top of the stack)

b. **(5 marks)** Assume that the following sequence of operations were performed on the stack above:

```
pop();
push(4);
push(2);
pop();
pop();
push(-7);
pop();
```

Show the array after the above operations.

SOLUTION

`pop();`	5, 3, 17, -1, -9
`push(4);`	5, 3, 17, -1, -9, 4
`push(2);`	5, 3, 17, -1, -9, 4, 2
`pop();`	5, 3, 17, -1, -9 ,4
`pop();`	5, 3, 17, -1, -9
`push(-7);`	5, 3, 17, -1, -9, -7
`pop();`	5, 3, 17, -1, -9

Final stack:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value	5	3	17	-1	-9								