Introduction to AI

Assignment 1

February 21, 2023

1. Write True/False for the following conditional independence statements. Justify clearly your answer by showing active/blocked trails as necessary and appropriate rules for them to be active/blocked. [No coding required for this question. Each sub-question has **2 points**]

   (a) $A \perp G \mid \{F\}$
   False. There is a V-structure between A and G.

   $$A \rightarrow B \rightarrow D \leftarrow G$$

   Given F and thus implying we know D, it couples A and G. Thus A and G are dependent.

   (b) $A \perp G \mid \{E\}$
   False. There is a common cause structure.

   $$A \leftarrow C \rightarrow E \rightarrow G$$

   A and E are dependent, and thus by cascade structure A and G are also dependent. But given E, it blocks the active path between A and G. Thus A and G are independent given E.

   (c) $A \perp E \mid \{C\}$
   True. A, C and E have a common cause structure.

   $$A \leftarrow C \rightarrow E$$

   Given the common cause C, it decouples A and E. There are no other active paths between A and C, thus they are independent.

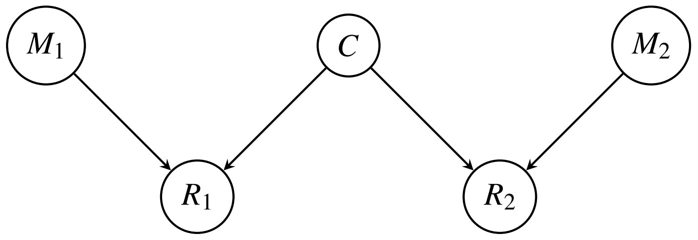
   (d) $A \perp E \mid \{C, D\}$
   False. Similar to part (c), given C it decouples A and E at $A \leftarrow C \rightarrow E$. However, there exists a V-structure between A and E.

   $$A \rightarrow B \rightarrow D \leftarrow E$$

   Similar to part (a), given D it couples A and E. Thus A and E are dependent.

   (e) $A \perp D \mid \{B, E\}$
   True. Cascade from $A \rightarrow B \rightarrow D$ is blocked given B. Common cause couples A and E. But path from $A \leftarrow C \rightarrow E \rightarrow D$ is blocked given E. Since all paths from A to D are blocked, A and D are independent.

2. (a) Draw the Bayes net corresponding to this setup.



$$\begin{tikzcd}
M_1 \arrow[dr] & & C \arrow[dl] \arrow[dr] & & M_2 \arrow[dl] \\
& R_1 & & R_2 &
\end{tikzcd}$$

Variable Name	Domain	Interpretation
C	$\{1, 0\}$	The actual health of a person. Either COVID positive 1 or negative 0.
M_1	$\{a, b, c\}$	The manufacturer of the first test kit. Where the company is a, b or c.
M_2	$\{a, b, c\}$	The manufacturer of the second test kit. Where the company is a, b or c.
R_1	$\{1, 0\}$	The result of the first test kit. Either positive 1 or negative 0.
R_2	$\{1, 0\}$	The result of the second test kit. Either positive 1 or negative 0.

This table can be interpreted as three independent events M_1, M_2 and C. The manufacturer of the two test kits received by a person and their actual health are independent, but they will determine the result of the test kit.

(b) Write conditional probabilities (numerical values) associated with each node of this Bayes net. As there are 5 variables, please specify one conditional probability table (CPT) for each variable.

$P(C = 0)$	$P(C = 1)$
0.7	0.3

$P(M_n = a)$	$P(M_n = b)$	$P(M_n = c)$
0.333	0.333	0.333

M_n	C	$P(R_n = 0 \mid M_n, C)$	$P(R_n = 1 \mid M_n, C)$
a	0	0.99	0.01
b	0	0.95	0.05
c	0	0.91	0.09
a	1	0.3	0.7
b	1	0.2	0.8
c	1	0.1	0.9

For values of $n = \{1, 2\}$ as each person has two test kits.

(c) Are the results of the two tests dependent or independent given the evidence that the Covid Status is known? Justify your answer.

There is a common cause structure $R_1 \leftarrow C \rightarrow R_2$ which couples R_1 and R_2. But given the COVID status C, it decouples R_1 and R_2. Thus they are independent.

(d) Assume you took both tests at home. After being tested twice in a matter of minutes, the first test was positive and the second negative. What is the probability that you actually have COVID19? Show your analytical computations.

Given that the first test was positive and the second test was negative, then

$$R_1 = 1$$
$$R_2 = 0$$

We are trying to solve for $P(C = 1 \mid R_1 = 1, R_2 = 0)$. Since R_1 and R_2 are conditionally independent on C,

$$\begin{aligned} P(C = 1 \mid R_1 = 1, R_2 = 0) &= \frac{P(R_1 = 1, R_2 = 0 \mid C = 1)P(C = 1)}{P(R_1 = 1, R_2 = 0)} \\ &= \frac{P(R_1 = 1 \mid C = 1)P(R_2 = 0 \mid C = 1)P(C = 1)}{P(R_1 = 1, R_2 = 0)} \end{aligned}$$

$$\begin{aligned} &P(C = c, M_1 = m_1, M_2 = m_2, R_1 = r_1, R_2 = r_2) \\ =&P(C = c) \cdot P(M_1 = m_1) \cdot P(M_2 = m_2) \cdot P(R_1 = r_1 \mid C, M_1) \cdot P(R_2 = r_2 \mid C, M_2) \end{aligned}$$

3. The following algorithm determines whether an element is within a sorted array.

```java
// parameterized version
public static boolean binarySearch(int[] data, int target, int low, int high) {
   if (low > high)
      return false; // interval empty; no match
   else {
      int mid = (low + high) / 2;
      if (target == data[mid])
         return true; // found a match
      else if (target < data[mid])
         return binarySearch(data, target, low, mid - 1); // recur left
      else
         return binarySearch(data, target, mid + 1, high); // recur right
      }
   }


// Demonstration of a public wrapper function with cleaner signature
public static boolean binarySearch(int[] data, int target) {
   return binarySearch(data, target, 0, data.length - 1);
}
```

Suppose that you are given the following array as input:
`data = [2, 5, 7, 8, 10, 19, 20, 23, 25, 30, 35, 36];`

(a) Draw the recursion trace for `binarySearch(data, 33)`.

(b) What is the Big O complexity of binary search above? Justify it.

(c) How would you make it even more efficient?

(d) Describe a modification that will return the index of the target element or -1 if not found.

(b) Binary search is $O(logn)$. After every recursive call, the problem halves in size. For an array of size n, we call `binarySearch()` on array of size $n, \frac{n}{2}, \frac{n}{4} \ldots$ because the high or low halves after each recursive call. The body of `binarySearch()` only contains primitive operations, hence we can conclude that the growth rate of `binarySearch()` is $O(logn)$.

(c) Don't know...

(d) Change the return type of binarySearch to int. When found a match, return the index of the target, when no match return -1.

```java
public static int binarySearch(int[] data, int target, int low, int high) {
   if (low > high)
      return -1; // interval empty; no match
   else {
      int mid = (low + high) / 2;
      if (target == data[mid])
         return mid; // found a match; return index of target element
```

```
        else if (target < data[mid])
            return binarySearch(data, target, low, mid - 1);
        else
            return binarySearch(data, target, mid + 1, high);
        }
    }
```

4. Suppose that you wish to solve puzzles of the following form:

$$\begin{aligned} fish &= bird + cat \\ fish + cat &= 4birds \\ fish + bird + cat &= 10 \end{aligned}$$

The task is to assign a unique integer between 0 to 9 to each variable fish, cat, or bird so that the above equations hold true.
You chance upon a recursive algorithm below described in pseudocode.

(a) Specify the arguments k, S, U to solve the puzzle involving *fish*, *cat*, and *bird* above.

$$\begin{aligned} k &= 3 \\ S &= [\,] \\ U &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \end{aligned}$$

(b) How would you test whether S is a configuration that solves the puzzle?
Let S be an array with values

$$S = [x, y, z]$$

We can then check if S satisfies the 3 equations, by mapping the values of S to *fish*, *bird* and *cat*.
Let

$$\begin{aligned} x &= fish \\ y &= bird \\ z &= cat \end{aligned}$$

If the equations

$$\begin{aligned} x &= y + z \\ x + z &= 4y \\ x + y + z &= 10 \end{aligned}$$

hold true, then S is a valid solution.

(c) What are the values of *fish*, *bird* and *cat* in this case?

$$\begin{aligned} fish &= 5 \\ bird &= 2 \\ cat &= 3 \end{aligned}$$

$$\begin{aligned} 5 &\equiv 2 + 3 \\ 5 + 3 &\equiv 4 \times 2 \\ 5 + 3 + 2 &\equiv 10 \end{aligned}$$

(d) What is the Big O complexity of `PuzzleSolve`?
For set U with a size of n, we have a total of $n!$ permutations. From the code, each iteration through U recursively calls the function. So the function will call itself n times on the first recursion, and $n(n-1)$ total times on the second and so forth.
The total number of function calls is $n!$ which is the total number of permutations of elements in set U. Since `PuzzleSolve` only contains primitive operations, we can conclude that the function is $O(n!)$

5. You are given the following recursive method:

```java
public static long negativeFibonacci(int n){
   if (n <= 1)
      return n;
   else
      return 0 - negativeFibonacci(n-2) - negativeFibonacci(n-1);
}
```

(a) What is the output of `negativeFibonacci(567)`? Let negative fibonacci function be $f(n)$.

$$f(n) = 0 - f(n-2) - f(n-1)$$

Using $n = 3$,

$$\begin{aligned} f(3) &= 0 - f(3-2) - f(3-1) \\ &= 0 - f(1) - f(2) \\ &= 0 - f(1) - (0 - f(2-2) - f(2-1)) \\ &= 0 - f(1) - (0 - f(0) - f(1)) \\ &= -f(1) + f(0) + f(1) \qquad (1) \\ &= f(0) \\ &= 0 \end{aligned}$$

We can see that the function $f(1)$ cancels itself out in (1). By repeated use of definition of the negative Fibonacci sequence, for each integer $n \geq 1$

$$\begin{aligned} f(n) &= 0 - f(n-2) - f(n-1) \\ &= 0 - f(n-2) - (0 - f(n-1-2) - f(n-1-1)) \\ &= 0 - f(n-2) - (0 - f(n-3) - f(n-2)) \\ &= 0 - f(n-2) + f(n-3) - f(n-2) \\ &= f(n-3) \end{aligned}$$

It can be seen that for each integer $n \geq 1$, $f(n) = f(n-3)$. This cascades till the base case $n \leq 1$, then $f(n) = n$. We have three possible base cases, where $n \equiv n \bmod 3$.

$$\begin{aligned} f(n) &= f(n-3) \\ &= f(n \bmod 3) \end{aligned}$$

$$f(n) = \begin{cases} 0 & \text{if } n \bmod 3 = 0, \\ 1 & \text{if } n \bmod 3 = 1, \\ -1 & \text{if } n \bmod 3 = 2, \end{cases}$$

Given $n = 567$,

$$\begin{aligned} f(n) &= f(567) \\ &= f(567 \bmod 3) \\ &= f(0) \\ &= 0 \end{aligned}$$

(b) What is the worst case complexity of `negativeFibonacci(567)`? Explain your answer.
The worst case complexity is $O(2^n)$. Although we have mathematically proven that $f(n)$ decomposes to $f(n-3)$, the function doesn't make use of that equivalence. Instead, it unconditionally calls itself twice per recursion. This is the same as the `fibonacciBad` in the lecture slides.
Proof by induction:

$$\begin{aligned} &\text{Let } g(n) \text{ be the number of additions.} \\ &\text{Let } P(n) : g(n) < c(2^n) \qquad \text{where } c > 0 \end{aligned}$$

Base case:

$$\begin{aligned} g(0) &= 0 < 2^0 = 1 \\ g(1) &= 0 < 2^1 = 2 \\ g(2) = 0 - g(0) - g(1) &= 0 < 2^2 = 4 \end{aligned}$$

Base case is true.
Inductive step: Assume $P(n)$ is true for $n-3$, $n-2$, $n-1$, for all $n \geq 3$,

$$g(n) = 0 - g(n-2) - g(n-1) \tag{1}$$
$$g(n-1) = 0 - g(n-3) - g(n-2) \tag{2}$$

From (1) and (2)

$$g(n-1) = g(n) + g(n-1) - g(n-3)$$
$$g(n) = g(n-3) \tag{3}$$

From assumption $P(n-3)$ and (3),

$$g(n-3) \leq 2^{n-3}$$

$$\begin{aligned} g(n) &= g(n-3) \\ &\leq 2^{n-3} \\ &= 2^n \cdot 2^{-3} \\ &= \frac{1}{8} \cdot 2^n \\ &\leq 2^n \end{aligned}$$

$$P(n-3) \longrightarrow P(n)$$

$$\begin{aligned}
P(3-3) &\longrightarrow P(0)\\
P(4-3) &\longrightarrow P(1)\\
P(5-3) &\longrightarrow P(2)
\end{aligned}$$

Hence $P(n)$ true for all $n \geq 0$.
The function `negativeFibonacci` is $O(2^n)$ where $c = 1$ and $n_0 = 1$.

6. For each of the following algorithm `sum1` to `sum4` below:

   (a) What is the output when $n = 10$ and $k = 5$?

   (b) Describe in terms of n and k what the output value will approach as the inputs grow.

   (c) What is the worst complexity?

   `//sum1`

```java
public static double sum1(int n){
    if (n == 0)
        return n;
    else
        return (double)n + sum1(n-1);
}
```

   In `sum1`, $f_1(n)$ can be defined recursively as such

$$f_1(n) = \begin{cases} 0 & \text{if } n = 0, \\ n + f_1(n-1) & \text{if } n > 0, \end{cases}$$

   `sum1` simply adds all numbers from n till 0.

$$\begin{aligned} f_1(n) &= n + (n-1) + (n-2) + \ldots + 1 + 0 \\ &= \frac{n(n+1)}{2} \end{aligned}$$

   When $n = 10$,

$$\begin{aligned} f_1(n) &= f_1(10) \\ &= 10 + 9 + \ldots + 1 + 0 \\ &= \frac{10(10+1)}{2} \\ &= 55 \end{aligned}$$

   Output value approaches $\frac{n(n+1)}{2}$ as n grows.
   Worst case complexity is $O(n)$.

```java
//sum2
public static double sum2(int n){
   if (n == 0)
      return n;
   else
      return (double)n + sum2(n/2);
}
```

In sum2, $f_2(n)$ can be defined recursively as such

$$f_2(n) = \begin{cases} 0 & \text{if } n = 0, \\ n + f_2\left(\left\lfloor \frac{n}{2} \right\rfloor\right) & \text{if } n > 0, \end{cases}$$

When $n = 10$,

$$\begin{aligned} f_2(10) &= 10 + f_2\left(\frac{10}{2}\right) \\ &= 10 + f_2(5) \\ &= 10 + 5 + f_2(2) \\ &= 10 + 5 + 2 + 1 + 0 \\ &= 18 \end{aligned}$$

The function adds half of its input value for every recursive call, until the input becomes 0. Let a be the largest integer such that $\frac{n}{2^a} \geq 1$. Then $a + 1$ is the number of iterations of sum2, while ignoring the floor function.

$$\begin{aligned} f_2(n) &\approx n + \frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^a} \\ &\approx n + (n - 1) \\ &= 2n - 1 \\ \frac{n}{2^a} &\geq 1 \\ n &\geq 2^a \\ \log n &\geq a \log 2 \\ a &\leq \log n \end{aligned}$$

The worst case complexity is $O(\log n)$. The input value n halves on every recursion.

```java
//sum3
public static double sum3(int n, int k){
    if (k == 0)
        return n;
    else
        return (double)n + sum3(n, k-1);
}
```

In `sum3`, $f_3(n, k)$ can be defined recursively as such

$$f_3(n,k) = \begin{cases} n & \text{if } \ k = 0, \\ n + f_3(n, \ k-1) & \text{if } \ k > 0, \end{cases}$$

We can see that the number of additions is $k + 1$.

$$\begin{aligned} &\text{Adds } k+1 \text{ times} \\ f_3(n,k) &= n + n + \ldots + n \\ &= (k+1)n \\ &= nk + n \end{aligned}$$

When $n = 10$ and $k = 5$,

$$\begin{aligned} f_3(10,5) &= 10(5) + 10 \\ &= 60 \end{aligned}$$

The worst case complexity is $O(k)$.

```java
//sum4
```

```java
public static double sum4(int n, int k){
   if (k == 0)
      return n;
   else
      return (double)n + sum4(n, k/2);
}
```

In sum3, $f_4(n,k)$ can be defined recursively as such

$$f_4(n,k) = \begin{cases} n & \text{if } \ k = 0, \\ n + f_4\left(n, \left\lfloor \frac{k}{2} \right\rfloor\right) & \text{if } \ k > 0, \end{cases}$$

Function sum4 adds n for some a number of times. Then
when k=6

Let a be the number of additions in sum4.

$$\begin{aligned} f_4(n,6) &= n + f_4\left(n, \left\lfloor \frac{6}{2} \right\rfloor\right) \\ &= n + f_4(n,3) \\ &= n + n + f_4\left(n, \left\lfloor \frac{3}{2} \right\rfloor\right) \\ &= n + n + f_4(n,1) \\ &= n + n + n + f_4\left(n, \left\lfloor \frac{1}{2} \right\rfloor\right) \\ &= n + n + n + f_4(n,0) \\ &= n + n + n + n \\ &= 4n \\ a &= 4 \end{aligned}$$

Then it can be seen that

$$\begin{aligned} a &= 1 + \lfloor \log_2 k \rfloor + 1 \\ &= 2 + \lfloor \log_2 k \rfloor \end{aligned}$$

Where $\lfloor \log_2 k \rfloor$ is the maximum number of times k can be divided by 2 before $k \leq 1$. We add 2 for the n in the first recursion and for the n in the base case.
Therefore

$$\begin{aligned} f_4(n,k) &= an \\ &= (2 + \lfloor \log_2 k \rfloor)n \\ &= 2n + \lfloor \log_2 k \rfloor n \end{aligned}$$

When $n = 10$ and $k = 5$,

$$\begin{aligned} f_4(10,5) &= 2(10) + \left\lfloor \log_2 5 \right\rfloor (10) \\ &= 20 + \lfloor 2.321928095 \rfloor (10) \\ &= 20 + 20 \\ &= 40 \end{aligned}$$

The worst case complexity is $O(\log n)$, as the input n halves on every recursion.

7. Describe a way to use recursion to compute the sum of all the elements in an $n \times n$ (two dimensional) array of integers. What is the complexity?

```java
//Sum 1D array
public static int sumX(int x, int[] arr){
    if(x == 0)
        return arr[0];
    else
        return arr[x] + sumX(x-1, arr);
}

//Sum 2D array
public static int sumAll(int y, int[][] arr, int n){
    if(y == 0)
        return sumX(n-1, arr[0]);
    else
        return sumX(n-1, arr[y]) + sumAll(y-1, arr, n);
}
```

The complexity is $O(n^2)$. The code is analagous to a nested for loop.