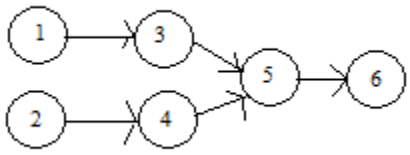
Past COP 3503 Final Exam Solutions

1) (Spr 18) Below is a 6 line code segment in Python that prompts the user to enter the distance and average speed of a trip and then calculates the time the trip took. For these lines of code, find each direct dependency (ie line 3 must go before line 5) necessary for the code to run properly. Draw the corresponding graph using each line as a vertex and each dependency as an edge. Then, count the number of possible topological sorts of this graph and explain the significance of this number.

```
lemons = int(input("How many lemons?\n"))                    # line 1
sugar = int(input("How many cups of sugar?\n"))              # line 2
limitLemons = lemons//3                                      # line 3
limitSugar = sugar*4                                         # line 4
pitchers = min(limitLemons, limitSugar)                      # line 5
print("You can make",pitchers,"pitchers of lemonade.")       # line 6
```

Graph



Number of Top Sorts: **6**

Significance of # of top sorts: **Number of different orders the lines of code above could be ordered to produce a correct program.**

2) (Sum 14) Dynamic Programming (Matrix Chain Multiplication)

Using the dynamic programming algorithm shown in class, determine the minimum number of multiplications to calculate the matrix product ABCD, where the dimensions of A, B, C and D are given below:

A: 2 x 5, B: 5 x 4, C: 4 x 1, D: 1 x 5

	A	B	C	D
A	0	**40**	**30**	**40**
B	X	0	**20**	**45**
C	X	X	0	**20**
D	X	X	X	0

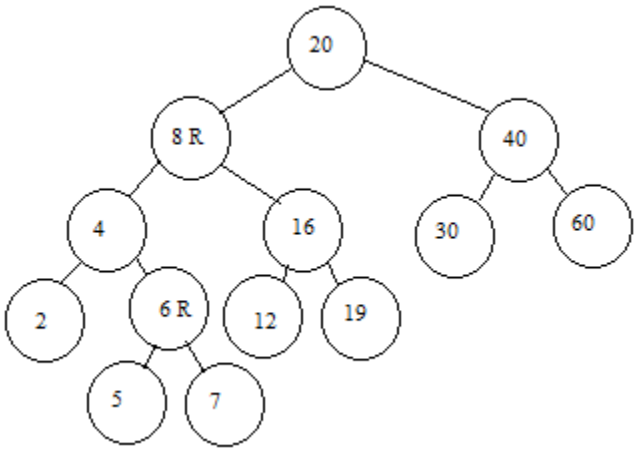
First diagonal: 3 pts total

(AB)C: 40 + 0 + 2x4x1 = 48 **A(BC): 0 + 20 + 2x5x1 = 30**

(BC)D: 20 + 0 + 5x1x5 = 45 **B(CD): 0 + 20 + 5x4x5 = 120**

(ABC)D: 30 + 0 + 2x1x5 = 40 **(AB)(CD): 40 + 20 + 2x4x5 =100**
A(BCD): 0 + 45 + 2x5x5 = 95

3) (Spr 18) Draw the result of deleting the value 40 from the red-black tree shown below. In the drawing below, red nodes are indicated with a letter 'R' next to the number stored in the node. In your solution, please put an 'R' in each node that is red. (Note: use the regular binary tree rules for deleting a value which is stored in a node with 2 children.) If you explain your thinking, you may get partial credit for incorrect solutions.



Note: When deleting 40, we can either delete the physical location of node 30 or 60. Due to the length of the solution, this solution only shows the result of deleting the physical

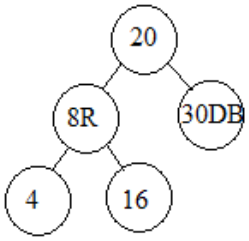
location of the 30. A second correct answer exists (and was properly graded for) where the physical node storing 60 is deleted.

After we denote the physical node storing 30 to be deleted, we copy 30 into the node that originally stored 40. Here is the state of affairs from that point, just for the subtree rooted at 30:

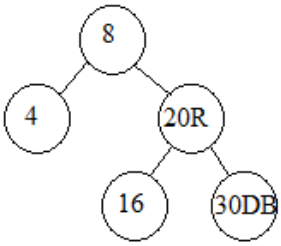
Initially, the null node is a double black.

We push the double black up one level.

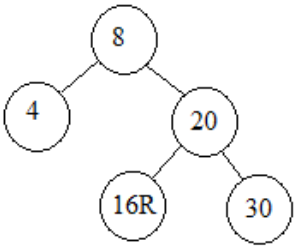
Then, we deal with a double black that has a red sibling:



This is the original picture.

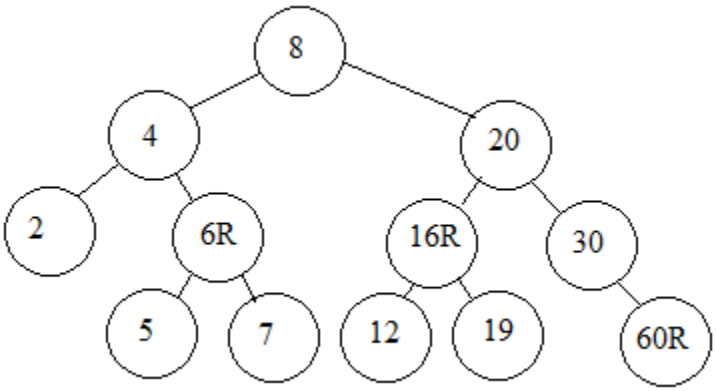


Here is when we right rotate making 8 the root and 20 red.



Push the double black up one level, fixing the issue.

Here is the final tree after those changes are made:



Note: if we had deleted the physical node storing 60 instead of the one storing 30, the only difference in the final answer is that 60 would be a black child of 20 and 30 would be a red left child of 60.

4) (Sum 14) Imagine testing 817 for primality using the Miller-Rabin primality test. In a single test, you'll successively calculate a randomly chosen base a raised to various powers mod 817. What are each of those powers?

n - 1 = 817 - 1 = 816. This is divisible by 16: 816 = 16 x 51. Thus, the list of powers are:

51, 102, 204, 408 and 816, respectively.

5) (Spr 18) The country of PaperTrailLandia runs its national presidential election via a very simple paper voting system. Each citizen submits a single piece of paper with the name of a single person, for whom they wish to vote. For the purposes of this problem, we assume that each name is a string of uppercase letters and that each distinct person has a distinct name. Complete the method below which takes all pieces of paper as a String array, and returns a HashMap<String,Integer> which maps each person receiving a vote to the number of votes they received.

```
public static HashMap<String,Integer> getMap(String[] names) {

    HashMap<String,Integer> map = new HashMap<String,Integer>();

    for (int i=0; i< names.length; i++) {

        if ( map.containsKey(names[i]) )

            map.put(names[i], map.get(names[i]) + 1);

        else

            map.put(names[i], 1);

    }

    return map;
}
```

6) (Sum 14) Dynamic Programming – Longest Increasing Sequence

Assume that you've already written a method LCS, that calculates the length of the longest common subsequence between two sequences of integers. (Its prototype is given below.) Write a method that takes in 1 sequence of integers and calculates its longest increasing sequence. For ease of implementation, assume that the input only contains distinct integers. Note: You may use both the Arrays.copyOf and Arrays.sort methods. These are listed below. (Note: The code is pretty short.)

```
// Returns an array storing the first newLength elements of original.
int[] copyOf(int[] original, int newLength);

// Sorts the specified array into ascending numeric order.
void sort(int[] a);
```

```
public static int lis(int[] seq) {
    int[] sorted = Arrays.copyOf(seq, seq.length);
    Arrays.sort(sorted);
    return lcs(seq, sorted);
}

public static int lcs(int[] x, int[] y) {

    int[][] table = new int[x.length+1][y.length+1];

    for (int i = 1; i<=x.length; i++) {
        for (int j = 1; j<=y.length; j++) {
            if (x[i-1] == y[j-1])
                table[i][j] = 1+table[i-1][j-1];
            else
                table[i][j] = Math.max(table[i][j-1], table[i-1][j]);
        }
    }
    return table[x.length][y.length];
}
```

7) (Sum 14) Dynamic Programming – Zero/One Knapsack Problem

Find the maximum valued knapsack of size 12 or less choosing from the following items (only 1 copy of each item is available). To get credit, please use the algorithm shown in class. Here are the items from which you are choosing:

Item	Weight	Value
Apple	3	4
Banana	4	6
Cantaloupe	7	13
Durian	5	9
Emblic	2	5
Fig	1	3

Show the algorithm by filling in the following table:

Item	1	2	3	4	5	6	7	8	9	10	11	12
Apple	**0**	**0**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
Banana	**0**	**0**	**4**	**6**	**6**	**6**	**10**	**10**	**10**	**10**	**10**	**10**
Cantaloupe	**0**	**0**	**4**	**6**	**6**	**6**	**13**	**13**	**13**	**17**	**19**	**19**
Durian	**0**	**0**	**4**	**6**	**9**	**9**	**13**	**13**	**15**	**17**	**19**	**22**
Emblic	**0**	**5**	**5**	**6**	**9**	**11**	**14**	**14**	**18**	**18**	**20**	**22**
Fig	**3**	**5**	**8**	**8**	**9**	**12**	**14**	**17**	**18**	**21**	**21**	**23**

8) (Spr 18) In class, a dynamic programming algorithm was covered that efficiently determines the ***fewest*** number of coins necessary to make change for a particular number of cents given the valid denominations of coins (and an infinite supply of each). In this algorithm, the array entry for dp[value] simply stores the fewest number of coins necessary to make change for value number of cents. For this problem, fill in this array for indexes 1 to 22, given that the valid denominations of coins are 1 cent, 3 cents, 8 cents and 14 cents.

index	0	1	2	3	4	5	6	7	8	9	10	11
mincoins	0	**1**	**2**	**1**	**2**	**3**	**2**	**3**	**1**	**2**	**3**	**2**

index	12	13	14	15	16	17	18	19	20	21	22
mincoins	**3**	**4**	**1**	**2**	**2**	**2**	**3**	**3**	**3**	**4**	**2**

You are given n k-sided fair dice, each labeled 1, 2, 3, …, k. You roll all of them. Then, separate out the ones that show k. Take the rest and roll them all again. Then, separate out the ones of these that show k, and roll the rest again. Repeat this process until you've separated all the dice out (ie, they all show k). The question we want to analyze is the number of times we expect to roll before completing the game.

9) (Spr 18) One way to analyze this is to write a simulation, run it many times and take the average. The latter portion of this is fairly trivial, so for this question, you will simply write a single Java method that takes in n, the number of dice, and k, the number of sides on each dice, simulates this process, and returns the number of turns it took to complete a single simulation of the game. Fill in the method signature given below. Assume that you have access to a static class variable r, that is of type Random, which you can use to generate random integers.

```
public static Random r;

public static int numTurnsSim(int n, int k) {

     int turns = 0;
     while (n > 0) {

          int numK = 0;

          for (int i=0; i<n; i++) {
               int die = r.nextInt(k)+1;
               if (die == k) numK++;
          }

          turns++;
          n -= numK;
     }

     return turns;
}
```

10) (Spr 18) Let the dice have k sides each and let T(n) equal the expected number of turns to complete the simulation described previous. A recurrence relation that T(n) satisfies is as follows:

$$T(0) = 0$$

$$T(n) = 1 + \sum_{i=0}^{n} [\binom{n}{i} \left(\frac{1}{k}\right)^{i} \left(\frac{k-1}{k}\right)^{n-i} T(n-i)]$$

In words, explain why this formula is correct. In your explanation, please explain what 1 represents, what the summation index i, represents and what T(n-i) represents. (Of course, explain what each part represents and why their interaction is the way that it is.) Note: One issue with this formula is that a T(n) term appears on the right hand side. But, if one were to solve this recurrence, we can easily take care of this problem by subtracting that term to the left hand side and factoring out T(n). Then, we would have a factor by which we could divide both sides of the resulting equation.

The 1 represents a turn throwing the the n dice. Of these dice, any where from 0 to n of them could turn up to land showing k. The variable i in the summation index represents the number of dice that show k out of the n dice that are rolled. For each of these possibilities, we must calculate the probability of that happening and multiply that by the expected number of turns of the simulation which will now have n - i dice left. The probability that i dice show up with k is based on the binomial distribution. In general, if we repeat a trial n times where the probability of success is p, the probability of getting exactly i success is $\binom{n}{i}(p)^{i}(1-p)^{n-i}$. For this particular problem, the probability of success, ie. rolling a k, is $\frac{1}{k}$. This explains the first three terms in the sum; these are just the product that represents the probability that i of the dice will show k. We must multiply this probability by the expected number of future turns, but this is just T(n - i), since T(x) represents the expected number of turns in the game starting with x die for any non-negative integer x. If i of the dice show k, then that means we continue to play the game with n - i dice, and using the definition of expectation, it follows that the appropriate term to multiply the aforementioned probability by is just T(n - i). This explains the recurrence relation above in full.

11) (Sum 14) Graphs – Topological Sort

Alice has to complete items 1 through 10. The following ordered pairs show the dependency between the items she must complete. Namely, for each ordered pair (a, b) shown below, she must complete item a before item b.

(2, 7), (8, 3), (9, 2), (4, 5), (4, 1), (4, 7) and (6, 10).

Show the ordering of the items produced by the algorithm shown in class that builds the list from the front and always adds "safe" nodes iteratively. When choosing between multiple possible "safe" nodes, always add the lowest numbered one first.

4, 1, 5, 6, 8, 3, 9, 2, 7, 10 (Grading: 1/2 pt each all or nothing round down)

12) (Sum 14) Algorithm Design – Dynamic Programming

Describe a dynamic programming algorithm (in words) to solve the following problem:

Soccer players are ordered 0, 1, 2, …, n – 1, where n is a positive integers less than or equal to 100. Each player i ($0 \leq i \leq n-2$) can pass to any player j such that $j > i$. The probability the pass will succeed is prob[i][j]. (Assume that this information is given in the input.) As with all probabilities, note that $0 \leq prob[i][j] \leq 1$, for all $0 \leq i < j \leq n - 1$.

We are allowed to choose any sequences of passes in order to move the ball from player 0 to player n – 1. (This sequence will necessarily be some subsequence of 0, 1, 2, …, n – 1, since all the players can only pass the ball in "one direction.") If all players act optimally, design an algorithm that calculates the probability that player n – 1 successfully receives the ball.

Solution

We will successively calculate f(i), for increasing values of i, where f(i) represents the maximal probability of player i receiving the ball. (5 pts) We can store these values in an array. We initialize f(0) to 1. since this is where the ball starts. To calculate an arbitrary f(i), remember that we know the values of f(x), for all x < i. Since player i must get the ball on a direct pass from some player x, where $0 \leq x < i$, we can simply calculate each of these i probabilities and take the maximum. (3 pts) Thus, mathematically, we have:

f(i) = max$(f(x)prob[x][i])$, $0 \leq x < i$. (5 pts)

Thus, we simply calculate this value for each value of i from 1 to n - 1, in this order. (2 pts) The run time of our algorithm is $O(n^2)$.

Note: The solution described in class (4/21/22) does this in reverse and also correctly solves the problem.