COP 3503 Spring 2022 Section 1 Final Exam Solutions

Date: 4/28/2022

1) (10 pts) Show the order of edges added into the minimum spanning tree of the graph with the adjacency matrix shown below when running Prim's algorithm starting from vertex **C**. Make sure to also indicate edges that were considered but not added because they would have created a cycle with the previously added edges.

From/To	A	B	C	D	E	F
A	--	8	--	--	9	10
B	8	--	--	2	7	5
C	--	--	--	10	8	--
D	--	2	10	--	3	3
E	9	7	8	3	--	4
F	10	5	--	3	4	--

Please fill out the table below in the order that the edges get pulled out from the priority queue in consideration for being added. Just write both letters denoting the edge. For example, the edge connecting B and D should be written as BD. If the edge gets added to the MST, write "Yes" in the second column. Otherwise, write, "No". It's possible there are more rows than necessary on the chart below.

Edge to Consider	Added? (Yes/No)
CE	Yes
ED	Yes
DB	Yes
DF	Yes
FE	No
BF	No
BE	No
AB	Yes

Grading: 1 pt for each of the four edges and yes, 5 pts total for the no edges, 1 pt for the last yes edge.

2) (6 pts) Consider running Karatsuba's algorithm for multiplying two eight-bit integers, X and Y, shown below, by breaking both down into two four-bit parts. The algorithm entails computing three four-bit multiplications, one of which could have a tiny bit of overflow. For the following example, show the three recursive multiplications that would be calculated using Karatsuba's:

```
X = 01011101
Y = 10110110
```

In each of the slots below, write **in binary** the two numbers that would have to be multiplied in the three recursive calls. Product #1 is the one that may end up having 5-bit number(s) in it. (Hint: The only actual computation that needs to be done in this problem is a tiny bit of binary addition.)

P1 = (0101 + 1101) x (1011 + 0110) = 10010 x 10001 (5 + 13 = 18, 11 + 6 = 17)
P2 = 0101 x 1011 (high bits) P3 = 1101 x 0110 (low bits)

Product #1: **10010** x **10001**
Product #2: **0101** x **1011**
Product #3: **1101** x **0110**

Grading: 1 pt per term. Products 2 and 3 can be swapped and the order of the terms in each product can be swapped.

3) (7 pts) Consider counting the number of ways to make change for 14 cents using 1 cent, 3 cent, 5 cent and 8 cent coins. Fill in the table below showing the work that the dynamic programming algorithm to solve the problem would do to solve the problem. The first row has been filled out for you for convenience.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	2	2	2	3	3	3	4	4	4	5	5	5
5	1	1	2	2	3	4	4	5	6	7	8	9	10	11
8	1	1	2	2	3	4	4	6	7	8	10	11	13	15

Grading: 2 pts for the first row, 2 pts for the second row, 3 pts for the last row. To get full points on a row, all values must be correct. To earn 1 pt on the first two rows, more than half have to be correct. To earn 1 on the last row 4 have to be correct, to earn two, 8 have to be.

4) (5 pts) What is the fewest number of comparisons necessary to sort 11 numbers via a comparison sort? Note: 11! = 39,916,800 and $2^{23} = 8{,}388{,}608$.

We want the minimum integral value of k such that $2^k > 11!$
Notice that $2^{25} > 32{,}000{,}000$ and $2^{25} < 36{,}000{,}000$. (Multiply 8 or 9 million by 4 to bound.)
It follows that $2^{26} > 64{,}000{,}000 > 11!$ while $2^{25} < 36{,}000{,}000 < 11!$. Thus, it follows that the desired fewest number of comparisons is **26.**

Grading: 2 pts for $2^k > 11!$ (otherwise 0 total), 3 pts to get to answer (give partial as needed)

5) (9 pts) Show the result of running the dynamic programming algorithm to find the edit distance between the strings "ACGATTACGA" and "CTAGACTTGA" in the table below. The first row has been filled in for you.

	C	T	A	G	A	C	T	T	G	A
A	1	2	2	3	4	5	6	7	8	9
C	1	2	3	3	4	4	5	6	7	8
G	2	2	3	3	4	5	5	6	6	7
A	3	3	2	3	3	4	5	6	7	6
T	4	3	3	3	4	4	4	5	6	7
T	5	4	4	4	4	5	4	4	5	6
A	6	5	4	5	4	5	5	5	5	5
C	7	6	5	5	5	4	5	6	6	6
G	8	7	6	5	6	5	5	6	6	7
A	9	8	7	6	5	6	6	6	7	6

Grading: 1 pt per row, must get the whole row to get the point.

6) (6 pts) The array below is the completed path array from running Floyd Warshall's Algorithm on a graph with vertices labeled 0 through 6. Using this array, write down the requested shortest paths. To write down a shortest path from vertex a to vertex b, write a sequence of 2 or more vertices, starting with a, ending with b, where each pair of consecutive vertices represents an edge on the shortest path between a and b. (Note: the array headers (indexes) are in bold and the contents of the array are not.)

From/To	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0	0	2	3	4	0	2	5
1	6	1	3	1	0	2	5
2	6	2	2	1	0	2	5
3	6	2	3	3	0	2	5
4	6	2	3	4	4	2	5
5	6	2	3	6	0	5	5
6	6	2	3	6	0	2	6

Note: You may not use all the slots provided.

Shortest Path from 0 → 6: **0, 4, 3, 2, 5, 6**

Shortest Path from 5 → 4: **5, 6, 0, 4**

Shortest Path from 1 → 0: **1, 3, 2, 5, 6, 0**

Grading: 2 pts per path, give 1 pt if start and end are correct and at least one intermediate vertex is in path in the right order.

7) (8 pts) Consider the following problem:

Given a string ***s*** and another string (text) ***t,*** determine the length of the longest prefix of ***s*** which is ALSO a substring of ***t.*** For example, if ***s*** = `racecar`, and ***t*** = `whereweracechance`, then the answer to the problem is 5, since the substring "`racec`" is contained in ***t***, but the substring "`raceca`" is not. Let ***m*** be the length of the string ***s*** and let ***n*** be the length of the string ***t.*** You may assume that ***m*** < ***n***.

In class on April 19th (the second to last day), we learned of a probabilistic technique to find if one string is a substring of another string. Using this technique with an adaptation, in words, describe a detailed algorithm to solve this problem in O(***nlgm***) expected time. Cleary describe your algorithm **and explain why the run-time is O(*nlgm*).**

The key observation is that if there is a prefix of s of length k that appears as a substring of t, then all other prefixes of length of s that are of a shorter length (than k) automatically also appear as substrings of t.

Thus, this property is binary searchable!

In class, we learned the rolling hashing technique to determine if a string s appears as a substring in another string t in O(n) time, where n is the length of the string t. For the purposes of answering this question and getting full credit, it's not necessary to explain the details of this method, just that we did it in class and its run time. For completeness, posted with the solutions is a code file called maxsubstring.java which solves this problem and includes the code that runs the hashing solution. (Note: This code isn't probabilistic because it verifies any matches found by hashing. The run time is an expected run time instead of a worst case run time.)

Thus, the algorithm is as follows:

1. Set low = 0, high = m.

2. while (low < high)
    a. mid = (low+high+1)/2
    b. Use hashing technique to see if s[0..mid-1] is a substring of t. (Verify matches.)
    c. if the answer to (b) is yes, set low = mid, else set high = mid-1.

3. Return low.

The for loop runs log(m) times because each time the difference between low and high gets divided by 2, so it runs at most k times where 2^k = m. Solving, we get k = log m. Steps 2a and 2b run in O(1) time, and as previously mentioned, step 2b takes O(n) time. It follows that the loop body has a run time of O(n) and the total run time is O(nlgm), as desired.

Grading: 4 pts binary search idea, 2 pts apply hashing idea, 2 pts for run time analysis.

8) (10 pts) Consider the following game: You start with ***n*** marbles, where ***n*** is a positive integer. The game ends when you have no marbles left. In a single turn, a random number of marbles in between 0 and ***n,*** inclusive, are taken from you. (Each of these ***n***+1 choices is equally likely to occur.) You continue taking turns until you have no marbles left. Let T(***n***) represent the expected number of turns the game should take if you start with ***n*** marbles.

(a) (3 pts) Write down a recurrence relation that T(n) satisfies. (Note: your answer should have a summation in it.) In writing this down, do not do any simplification; just use the definition about how a single turn works in the game. (The initial condition/base case is T(0) = 0.)

$$T(n) = 1 + \sum_{i=0}^{n} \frac{1}{n+1} T(i)$$

Grading: 1 pt for 1, 1 pt for sum and bounds, 1 pt for part inside the sum. Note: It's okay if the 1/(n+1) is factored out of the sum…and/or if the sum starts at 1.

(b) (7 pts) The equation from part (a) has the term T(n) appearing on both the left and right hand sides of the equation. These two terms can be combined so that T(n) appears on the left hand side only with terms ranging from T(0) to T(n-1) appearing on the right hand side. Also, noticing that T(0) = 0, T(0) can be removed. Do the algebra so that you have an expression for T(n) in terms of terms of the form T(i), where $1 \le i \le n-1$. (Note: when you do this, the form of your answer should be $T(n) = \frac{a}{b} + \frac{c}{d}\sum_{i=1}^{n-1} T(i)$, where a, b, c and d are relatively simple expressions, potentially in terms of ***n***.)

$$T(n) = 1 + \sum_{i=0}^{n} \frac{1}{n+1} T(i)$$

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

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

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

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

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

Thus, a = n+1, b = n, c = 1 and d = n (in fitting the form in the note.)

Grading: 1 pt split off term, 2 pts subtract to LHS, 2 pts factor out, 1 pt mult (n+1)/n, 1 pt simplify into the desired form.

9) (10 pts) Write a static method in Java, **<u>using the iterative dynamic programming method</u>,** that takes in a non-negative integer, ***n***, and returns the value of T(n) from question 8. (Note: Yes, I am well aware that a correct answer on #8 is necessary to have a chance to solve this problem correctly!!! Also, as a hint, the run-time of your code should be $O(n^2)$.)

```
public static double t(int n) {

      // 1 pt
      double[] dp = new double[n+1];

      // 1 pt.
      for (int i=1; i<=n; i++) {

            // 2 pts - can be added later.
            dp[i] = 1.0*(i+1)/i;

            // 2 pts loop
            for (int j=0; j<i; j++)

                  // 3 pts
                  dp[i] += (dp[j]/i);
      }

      // 1 pt
      return dp[n];
}
```

Note: It turns out that if you do further math, you can prove that $T(n) = 1 + \sum_{i=1}^{n} \frac{1}{i}$, so further math can be used to improve the run-time of this computation from $O(n^2)$ to O(n). Here is the simplification from the last step of problem 8:

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

Let X(n) = $\sum_{i=0}^{n} \frac{1}{n+1} T(i)$. It follows that X(n-1) = $\left(\frac{1}{n}\sum_{i=1}^{n-1} T(i)\right)$. But recall that T(n) = 1 + X(n). So we get:

$$1 + X(n) = \frac{n+1}{n} + X(n-1)$$

$$X(n) = \frac{n+1}{n} - \frac{n}{n} + X(n-1)$$

$$X(n) = X(n-1) + \frac{1}{n}$$

For any recurrence of the form Z(n) = Z(n-1) + f(n) has the solution $Z(n) = Z(0) + \sum_{i=1}^{n} f(i)$. Since X(0) = 0, it follows that $X(n) = \sum_{i=1}^{n} \frac{1}{i}$. Thus, we can conclude that $T(n) = 1 + \sum_{i=1}^{n} \frac{1}{i}$.

What's even more awesome is that a student figured all of this out during the exam =)

Translating this to code we get an alternate accepted solution:

```
public static double t(int n) {

	// 1 pt
	double res = 1.0;

	// 3 pts
	for (int i=1; i<=n; i++)

		// 5 pts
		res += 1.0/i;

	// 1 pt
	return res;
}
```

10) (6 pts) Write another static method that simulates one marble game. This method also takes in a non-negative integer ***n*,** simulates the game via a random object (assume a static variable belongs to the class that is shown below), and returns the number of turns the game took for the simulation. Note: Assume that r has already been instantiated in main before playGame has been called.

```
public static Random r;

public static int playGame(int n) {

      // 1 pt
      int res = 0;

      // 1 pt
      while (n > 0) {
            res++;                          // 1 pt
            int sub = r.nextInt(n+1);       // 1 pt
            n -= sub;                       // 1 pt
      }

      return res;                           // 1 pt

}
```

11) (6 pts) Using the Master Theorem, write down the solution to the three recurrence relations shown below. (You should write down T(n) = O(?), where ? is a some function of n.)

(a) $T(n) = 4T\left(\frac{n}{2}\right) + O(n)$ **$O(n^2)$**

(b) $T(n) = 3T\left(\frac{n}{2}\right) + O(n^2)$ **$O(n^2)$**

(c) $T(n) = 64T\left(\frac{n}{4}\right) + O(n^3)$ **$O(n^3 \lg n)$**

Grading: 2 pts all or nothing on each.

12) (6 pts) A common tactic to solve some type of problems is a technique called "Coordinate Compression". In this technique, we start with a set of input being a set of unique integers that aren't consecutive. (For example, the set could be {5, 2, 9, 16, 3, 12}.) For each number in the set, we would like to know its 0-based rank. When we sort the set, the order of the values is 2, 3, 5, 9, 12 and 16. The information we would like to be able to retrieve easily is the rank of any number in the list. For example, the rank of 2 is 0 and the rank of 12 is 4. To be able to retrieve this information, we would like to create a HashMap, mapping each number to its rank. In this example, the map would store 2 → 0, 3 → 1, 5 → 2, 9 → 3, 12 → 4 and 16 → 5. Write a static method that takes in an array of long values (assumed to be distinct) and returns a HashMap<Long,Integer> storing this mapping.

```
public static HashMap<Long,Integer> compress(long[] vals) {
      // 1 pt.
      Arrays.sort(vals);

      // 1 pt.
      HashMap<Long,Integer> res = new HashMap<Long,Integer>();

      // 1 pt.
      for (int i=0; i<vals.length; i++)

            // 2 pts.
            res.put(vals[i], i);

      // 1 pt.
      return res;
}
```

13) (5 pts) A disjoint set of n ≥ 3 elements, after being initialized as n separate sets has k union operations run on it where each union operation is run on a distinct pair of items. After these k operations, the disjoint set is more than one separate set. What is the maximum value of k, in terms of n? (For a simple example, when n = 3, the value of k = 1. Once we do a union between any 2 of the 3 items, the next union is forced to combine all three items into one set.)

To create two separate sets, we could have 1 item in one set and n-1 items in the other. Since each operation is unique, we can limit ourselves to n-1 items in a set and still do $\binom{n-1}{2}$ union operations between every possible pair of items in any subset of n-1 items. Thus, the maximum value of k is $k = \binom{n-1}{2} = \frac{(n-1)(n-2)}{2}$. Another way to reason this out is to note that at least one item must NOT be unioned with the other n-1. Thus, there are n-1 possible union operations that can't be made out of the total $\binom{n}{2}$ possible union operations. Thus, the answer can also be computed as $\binom{n}{2} - (n-1) = \frac{n(n-1)}{2} - \frac{2(n-1)}{2} = \frac{(n-1)(n-2)}{2}$, as desired.

Grading: 5 pts for any correct answer regardless of justification. Give partial as needed for any incorrect answer based on logical steps.

14) (5 pts) In programming assignment #4 (Maze Magic), a regular breadth first search did not earn full credit because it would take too long on a select few cases. Explain why the fact a BFS runs in O(V+E) time on a graph with V vertices and E edges does NOT contradict the fact that for this program a BFS did not run sufficiently fast on a few cases. (Recall in the assignment that there were up to 1000 rows and 1000 columns and there were some teleportation squares.)

In the problem, there could be many copies of a single teleportation letter. The underlying graph would have to connect all copies of the same teleportation letter. Thus, if a maze had 200,000 A's, then that means the underlying graph has roughly 2,000,000,000 edges just to connect each square containing an 'A'. Thus, even with a run time of O(V + E), the reason the algorithm runs too slowly is that if a teleportation letter appears a lot, the number of edges in the corresponding graph is extremely high, much more than the 100,000,000 simple operations limit (for run time) frequently cited in class. More accurately, with there being n = 1000 x 1000 = 1,000,000 vertices, there could be a total of 5×10^{11} edges total, if almost all the squares have the same teleportation letter.

Grading: Key is mentioning the n^2 blowup of edges. Any clear mention of this gets full credit. Decide partial as you see fit.

15) (1 pt) Today is National Blueberry Pie Day. What fruit is typically an ingredient in blueberry pie?

Blueberries! (Give to all)