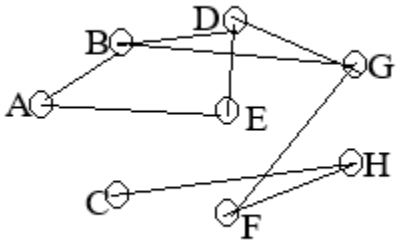
Fall 2026 COP 3503H Exam #2
Date: 11/5/2026

Name: _____

1) (15 pts) To dissuade students from flippantly signing up for the Bar Exam, the National Conference of Bar Examiners is instituting an interesting fee. For one's n^{th} attempt at passing the bar, the cost to take the exam is $(\$1000)n^2$. Sheila is determined to pass the Bar Exam and will continue taking it until she passes. Her probability of passing the exam on a single attempt is p. (Unfortunately, extra studying does not change the probability she passes the exam and each exam attempt is independent of the others.) **What is the expected amount of money Sheila will have to pay, in terms of p, to pass the bar?**

2) (8 pts) Show the result of running both a Depth First Search and Breadth First Search on the graph drawn below, starting at vertex A. Whenever either recursing or adding items to a queue, do so in alphabetical order.



DFS: _____ , _____ , _____ , _____ , _____ , _____ , _____ , _____

BFS: _____ , _____ , _____ , _____ , _____ , _____ , _____ , _____

3) (7 pts) Tommy must do the following five items today:

a) Deliver two packages to his mom.
b) Pick up a package from his dad.
c) Take an exam.
d) Study for an exam.
e) Pick up a package from his sister.

He has the following restrictions on the order in which he completes these tasks:

1. He must complete b before a.
2. He must complete e before a.
3. He must complete d before c.

In how many different orders can Tommy complete these five tasks within the set of restrictions? Please show work to justify your answer.

_____

4) (5 pts) The Ford-Fulkerson algorithm uses DFS to find augmenting paths while the Edmunds-Karp Algorithm uses BFS to do so. How does Dinic's (or Dinitz's) algorithm work to find augmenting paths? Intuitively, why does this work faster in practice?

5) (14 pts) Show the full trace of Dijkstra's Algorithm on the graph with the edges shown below with starting vertex A. When you fill out the table, in the leftmost column, indicate the vertex that gets added into the set of vertices for which correct shortest distances have been computed, followed by the shortest distance (after considering that vertex) to all of the other vertices. The vertices are labeled with the letters A through H, inclusive and the graph is given as 8 lists of directed edges, each list storing the outgoing edges from a particular vertex. Each edge is listed as (u, v, w), where u is the starting vertex, v is the ending vertex and w is the weight of the edge.

A: (A, B, 4), (A, C, 10), (A, D, 5)
B: (B, E, 12), (B, F, 7)
C: (C, E, 9), (C, F, 2)
D: (D, C, 3), (D, F, 13), (D, G, 2)
E: (E, H, 3), (E, F, 4)
F: (F, H, 10)
G: (G, H, 19)
H: None

Vertex	A	B	C	D	E	F	G	H
Add to Set	0	∞	∞	∞	∞	∞	∞	∞
A								

6) (10 pts) Using the crucial fact about minimum spanning trees taught in class, Elise has come up with an alternate algorithm to solve the minimum spanning tree problem:

1. Split the input graph vertices into two sets A and B of (nearly) equal size, if the graph has 3 or more vertices. (For 1 vertex, the MST is empty, for 2 vertices, it's the edge between the two vertices.)
2. Recursively find the MST of the vertices in set A (using only the edges between vertices in set A and do the same for set B.
3. Consider all edges between set A and set B. Of these, add the minimum edges to the two recursively formed MSTs and return this as the MST for the whole graph.

This algorithm doesn't work. Generate a small counter-example showing a graph for which it fails. Explain why the failure of this algorithm doesn't contradict the crucial fact about minimum spanning trees taught in class.

7) (15 pts) Consider a 50 x 50 grid with red pieces and black pieces, representing two teams. Each square can be empty, have one red piece on it, or have one black piece on it. On a single move, a piece can move up, down, left or right by one square. If a red piece moves to a square with a black piece on it, the red piece captures the black piece. A red piece is not allowed to move to a square with another red piece on it. A **parallel move** is when any subset of the red pieces all move at the same time. During this move, each red piece may move to either a square that is currently open or a square that contains a black piece. (Thus, if two red pieces are adjacent to each other, neither is allowed to move to the other piece's square.) Describe an algorithm to determine the maximum number of black pieces that the red pieces can capture on a single parallel move. Explain why your algorithm works.

8) (15 pts) Given a 2^n by 2^n grid of cells, with the top left corner having the position (s_r, s_c), the top right corner having position (s_r, $s_c + 2^n - 1$) and the bottom right corner having the position ($s_r + 2^n - 1$, $s_c + 2^n - 1$), we can describe a "path" to any cell on the grid as follows:

For n > 1, break the grid up into four quadrants, each of size 2^{n-1} x 2^{n-1}. If the cell is in the top left quadrant, then the path starts with the character '0'. If the cell is in the top right quadrant, the path starts with the character '1'. If the cell is in the bottom left quadrant, the path starts with the character '2'. Finally, if the cell is in the botttom right quadrant, the path starts with '3'. For example, here is an 8 by 8 grid with the path for each cell written into it.

000	001	010	011	100	101	110	111
002	003	012	013	102	103	112	113
020	021	030	031	120	121	130	131
022	023	032	033	122	123	132	133
200	201	210	211	300	301	310	311
202	203	212	213	302	303	312	313
220	221	230	231	320	321	330	331
222	223	232	233	322	323	332	333

Write a method (using Divide and Conquer, so not iteratively), that takes in the value of n (guaranteed to be positive), the values of s_r and s_c, as well as a cell location on row r, column c, and returns a String storing the path to that cell location. (Please use base case n = 1.) **Assume that there will be no overflow issues with the int data type.**

```
public static String path(int n, int sR, int sC, int r, int c) {

}
```

9) (10 pts) Let the integer array, path, be such that path[i][j] stores the last vertex visited in a graph on the shortest path from vertex i to vertex j. Complete the static method below so that it takes in the path array (note: the length of this array is the number of vertices in the graph it corresponds to), a start vertex number s, and an end vertex number, e, and returns an ArrayList<Integer> storing the shortest path from vertex number s to vertex number e. You may assume that s and e are distinct and that the list your code returns must start with s and end with e and that such a path exists.

```
public static ArrayList<Integer> getPath(int[][] path, int s, int e) {

}
```

10) (1 pt) One of the examples of a BFS application shown in class was the 8-Puzzle. In this game, how many tiles are arranged on the game board?

_____

ArrayList

Constructor	Description
`ArrayList<E>()`	Creates an empty ArrayList.

Modifier and Type	Method	Description
`boolean`	`add(E e)`	Adds e to the end of this ArrayList
`void`	`add(int index, E e)`	Inserts e at index index in this ArrayList
`void`	`clear()`	Empties the contents of this ArrayList.
`E`	`get(int index)`	Returns the item at index index of this ArrayList.
`E`	`remove(int index)`	Removes and returns the item at index index in this ArrayList.
`boolean`	`remove(Object o)`	Returns the first occurrence of o in this ArrayList. Returns false if o isn't in the list.
`int`	`size()`	Returns the number of elements in this ArrayList.

Scratch Page – Please clearly mark any work you would like graded.