Northeastern University
CS5100 – Foundations of Artificial Intelligence
Fall 2022 Semester, Richard Hoshino

Problem Set #2

This Problem Set is due at **6PM on Wednesday October 5**, and will be submitted on Canvas.

This Problem Set will be marked out of 40, and is worth 8% of the final course grade. There are five problems, each worth eight marks.

Problems #1 and #2 and #3 are to be completed individually, while Problems #4 and #5 are to be completed in your assigned small groups. For the two group problems, you are strongly encouraged, though not required, to have a different lead author for each of parts (a), (b), and (c).

Please type (or neatly handwrite) your solutions on standard 8.5×11 paper, with your name(s) at the top of each solution. Ensure that you submit five *separate* PDF files on Canvas, one for each problem. Make sure you label your Problem Set #1 submissions appropriately - e.g. richard2-1.pdf, richard2-2.pdf, richard2-3.pdf, richard2-4.pdf, richard2-5.pdf.

Given that the last two problems are done in a group, your final two PDF files will be identical to some of your classmates. (For example, richard2-4.pdf might be identical to yvonne2-4.pdf and bethany2-4.pdf). This is completely fine, and enables you to have a record of all of your submitted work in this course.

While a solution must be absolutely perfect to receive full marks, I will be generous in awarding partial marks for incomplete solutions that demonstrate progress.

So that there is no ambiguity, there are two non-negotiable rules. A violation of either rule constitutes plagiarism and will result in you receiving an F for this course.

(a) If you meet with a classmate to discuss one of the three Individual Problems, the articulation of your thought process (i.e., what you submit to me), must be an individual activity, done in your own words, away from others. Please remember that the solution-writing process is where so much of your learning will occur in this course: much more than anything we do in class, and even more than the time you spend on solving the problems. Do not be surprised if it takes you 3 to 5 times as long to write up a solution than it takes you to actually solve the problem. (For me, as an academic researcher writing formal proofs for publications, my ratio is significantly higher!)

(b) This Problem Set has been designed to be challenging, because struggling through problems is how we learn best. Your educational experience is cheapened by going online and finding the solution to a problem; even using the Internet to look for a "small hint" is unacceptable. In return, I will be readily available during our optional problem-solving workshops on Monday morning, and upon request, I will post hints to any questions you have on our class Canvas Page.

Problem #1 – INDIVIDUAL

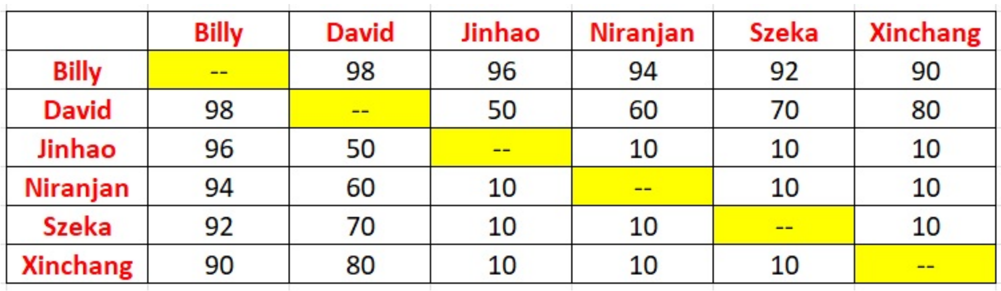
Consider a class of $n = 2m$ students, where m is a positive integer.

For each pair of students (i, j), let $C[i, j]$ be their *compatibility* score, which denotes how well they would work together. The highest possible compatibility score is 100 and the lowest possible compatibility score is 0. By definition, $C[i, j] = C[j, i]$.

Your task is to design an algorithm to partition these $2m$ students into m pairs, so that the sum total of the compatibility scores is as **large** as possible.

For example, if you divide $n = 6$ students with the pairs as $[1, 2], [3, 4], [5, 6]$, then the sum total is $C[1, 2] + C[3, 4] + C[5, 6]$.

(a) Suppose we have $n = 6$ students who have the following compatibility scores. Determine the optimal partition of these 6 students into 3 pairs.



	Billy	David	Jinhao	Niranjan	Szeka	Xinchang
Billy	--	98	96	94	92	90
David	98	--	50	60	70	80
Jinhao	96	50	--	10	10	10
Niranjan	94	60	10	--	10	10
Szeka	92	70	10	10	--	10
Xinchang	90	80	10	10	10	--

(b) Let $n = 2m$ be an arbitrarily large number. Design *three* different algorithms for solving this problem. For each algorithm, describe how your algorithm works, whether the output returns quickly or slowly, and whether the algorithm is *guaranteed* to return the optimal solution.

(c) On the Canvas Home Page, under "Notes from Class", you will see an Excel sheet containing a data set for $n = 100$ students, where each entry $C[i, j]$ denotes the compatibility score of students i and j.

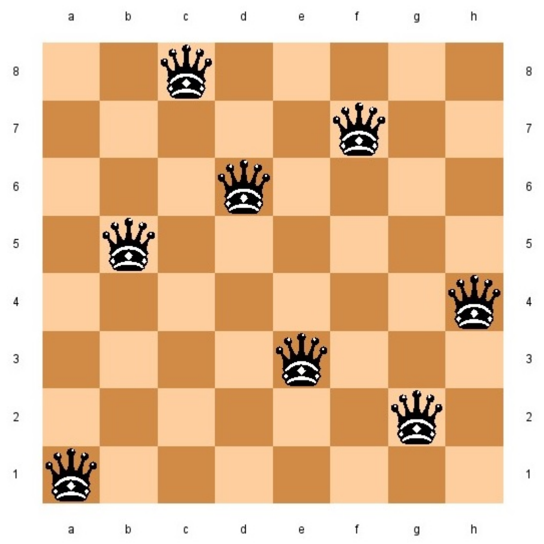
Using the data from this Excel sheet, apply one of your algorithms in part (b) to partition the $n = 100$ students into $m = 50$ pairs. Write a computer program to solve this problem, in the programming language of your choice.

For your solution to part (c), all you need to do is list your final total compatibility score. Submit your computer program as a separate file (.py, .java, .c, etc.)

You will receive full credit for any solution that is within 1% of the optimal solution, and partial credit otherwise.

Problem #2 – INDIVIDUAL

In the n-Queens problem, the goal is to place n queens on an $n \times n$ chessboard so that no pair of queens attacks each other, i.e., no pair of queens lies on the same row, column, or diagonal. For example, here is a solution to the 8-Queens problem.



(a) There are $4^4 = 256$ ways to place four Queens on a 4×4 chessboard, so that there is one Queen in each column. Determine how many of these 256 possibilities results in a solution to the 4-Queens problem, where no pair of Queens attack each other.

(b) Let's apply Hill-Climbing to attempt to solve the 5-Queens problem. Suppose our initial state is $(1, 2, 1, 2, 4)$, where the five Queens appear in cells $a1$, $b2$, $c1$, $d2$, $e4$. Use the Hill-Climbing search algorithm to improve this solution, and repeat this process until you cannot make any further improvements. Determine your final output, clearly showing all of your steps.

(c) Consider an $n \times n$ chessboard. Maximus and Minnie play the following adversarial game: on their turn, the player places a Queen anywhere on the chessboard, provided it does not attack a Queen that is already placed on the board. The first player who cannot move loses the game. Maximus moves first. Explain why Maximus has a winning strategy for the $n = 5$ game, and then explain why Maximus has a winning strategy for every n that is an *odd* number.

(d) **(OPTIONAL BONUS)** For a half-mark bonus, determine whether Maximus or Minnie has a winning strategy for the $n = 6$ game, and for an additional half-mark bonus, determine whether Maximus or Minnie has a winning strategy for the $n = 8$ game.

Problem #3 – INDIVIDUAL

Maximus and Minnie play a game of Tic-Tac-Toe, where Maximus plays X and Minnie plays O. Maximus goes first.

In this version of the game, Maximus and Minnie take turns, and the game is complete once all nine squares have been filled.

For each of the three rows, for each of the three columns, and for both of the diagonals, we award money as follows:

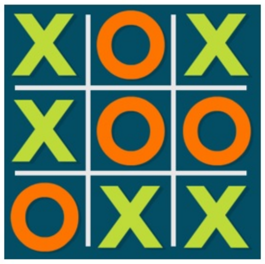
(i) If that row/column/diagonal has three X and zero O, then Maximus gets \$1000 from Minnie.

(ii) If that row/column/diagonal has two X and one O, then Maximus gets \$1 from Minnie.

(iii) If that row/column/diagonal has one X and two O, then Maximus loses \$1 dollar to Minnie.

(iv) If that row/column/diagonal has zero X and three O, then Maximus loses \$1000 to Minnie.

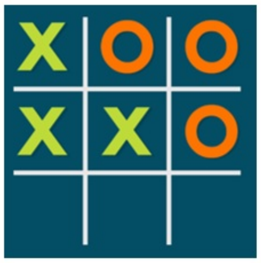
In the game below, Maximus's payoff is +2, since there are 5 rows/columns/diagonals with two X and one O, and 3 rows/columns/diagonals with one X and two O. So Minnie's payoff is −2.

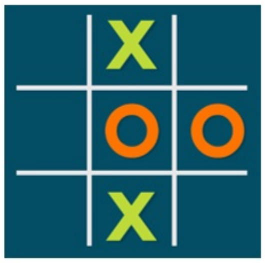


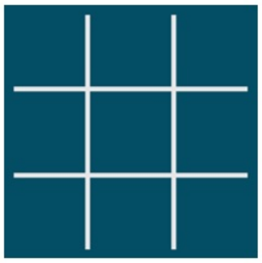
Naturally, both players wish to play to maximize their earnings, i.e., minimize their losses.

For each of the three boards below, it's Maximus's turn to move. Your task is to determine Maximus's *utility value* for each of these three boards, and use this result to determine the next move that Maximus should make.

In each of these three questions, feel free to use an adversarial search algorithm (e.g. Minimax, Alpha-Beta pruning) or reason it with pure logic. If you do the latter, make sure you rigorously argue each step, so that you are not missing any cases. If you decide to submit a computer program, make sure your code is well-documented so that the TA can follow your solution.







Problem #4 – GROUP

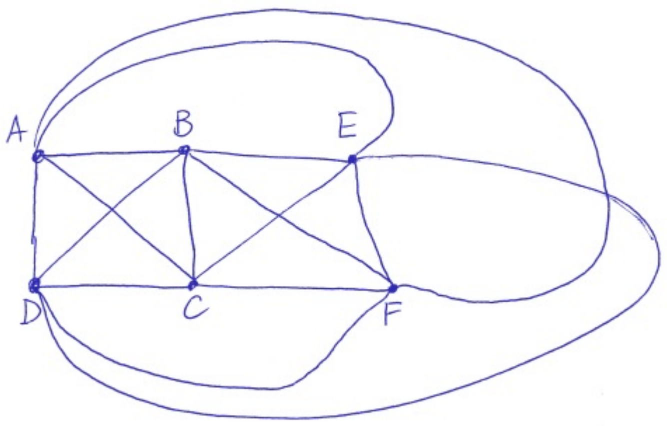
Let $n \geq 3$ be a positive integer. Consider a network of n cities, with each pair connected by a road. The Travelling Salesperson Problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"

Note that there is a simple and inefficient brute-force way to solve this problem. For example, in the case $n = 100$, we can check all $(99 \times 98 \times 97 \times \cdots \times 3 \times 2 \times 1)$ different ways we can connect the 100 cities in a Hamiltonian cycle, i.e., a Travelling Salesperson journey. Clearly this is not a smart approach for large values of n.

(a) Apply the Hill-Climbing Search Algorithm to solve the TSP, explaining how your algorithm would work for any value of n. Now apply your algorithm to the case $n = 6$ (see diagram below), starting with the randomly-chosen initial solution $AFBDECA$. Clearly describe the steps in your solution, showing how your algorithm improves the solution in each iteration. Is your final solution optimal?

(b) Use the A* search algorithm to determine a more efficient way to solve the TSP, explaining how your algorithm would work for any value of n. Make sure you explain how the A* search algorithm can be applied to the TSP: describe the states, the initial state, the goal state, the transition model, and the cost. Finally, generate a smart $h(n)$ function based on the minimum weight spanning tree. Explain why your heuristic function $h(n)$ is admissible, and why $h(n)$ is a good-enough estimate for the purposes of this problem.

(c) Apply your A* search algorithm to find the optimal TSP solution to the problem below, starting and ending with vertex A. Clearly describe your steps, and demonstrate that your (final) solution is indeed optimal.



In the figure above, which is definitely not to scale, we have the following edge weights:

AB=3, BC=4, CD=5, AD=6, BE=7, EF=8, CF=9,
AC=50, BD=50, CE=50, BF=50, AE=100, DF=100, AF=200, DE=200.

Problem #5 – GROUP

Maximus and Minnie play the same game of Tic-Tac-Toe as in Problem #3, where Maximus plays X and Minnie plays O. Maximus goes first. However this time Minnie's first move is **randomly chosen**, with each of the remaining squares equally likely to be chosen. Afterwards, she returns to being a rational agent, where she will play optimally.

The payoffs are the same as in Problem #3. For each of the three rows, for each of the three columns, and for both of the diagonals, we award money as follows:

(i) If that row/column/diagonal has three X and zero O, then Maximus gets $1000 from Minnie.

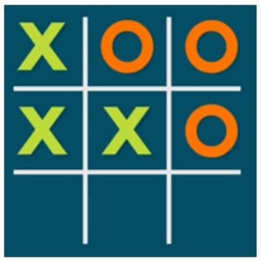
(ii) If that row/column/diagonal has two X and one O, then Maximus gets $1 from Minnie.

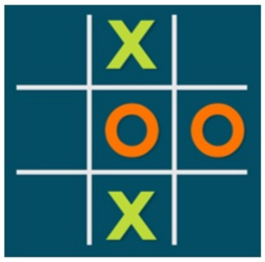
(iii) If that row/column/diagonal has one X and two O, then Maximus loses $1 dollar to Minnie.

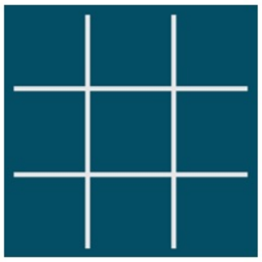
(iv) If that row/column/diagonal has zero X and three O, then Maximus loses $1000 to Minnie.

For each of the three boards below, it's Maximus's turn to move. Your task is to determine Maximus's *utility value* for each of these three boards, and use this result to determine the next move that Maximus should make.

In each of these three questions, feel free to use an adversarial search algorithm (e.g. Expectiminimax) or reason it with pure logic. If you do the latter, make sure you rigorously argue each step, so that you are not missing any cases. If you decide to submit a computer program, make sure your code is well-documented so that the TA can follow your solution.







NOTE: remember that Minnie's next move will be randomly chosen, and then after this random move, she will return to being a rational agent. So in the left board, Minnie's random move occurs on her fourth turn (the eighth move of the game). In the middle board, Minnie's random move occurs on her third turn (the sixth move of the game). And in the right board, Minnie's random move occurs on her first turn (the second move of the game).