Conditional Independence

Two events are independent of each other, given a third event.

$$P(A, B \mid C) = P(A \mid C)\, P(B \mid C)$$

Sentiment Analysis

S1 I love this movie — (+)

S2 This movie is amazing — (+)

S3 I hate this movie — (−)

S4 This movie is terrible — (−)

S5 I adore this film — (+)

"I love movie" — ?

Bag of words

Features	I	love	movie	'this'	'is'	amazing	hate	terrible	adore	film
S1 +	1	1	1	0	0	0	0	0		
S2 +	0	0	1	1	0	0	0	0		
S3 −	1	0	1	0	1	0	0	0		
S4 −	0	0	1	0	0	1	0	0		
S5 +	0	0	0	0	0	0	1	1		
→	1	1	1	1	1	1	1	1		

Objective : $P(+ \mid \text{'I love movie'})$

$P(- \mid \text{'I love movie'})$

$P(+) = \frac{3}{5}$

$P(-) = \frac{2}{5}$

$P(\text{love} \mid -) = 1/5+(2) = \frac{3}{5}$

$$P(+ \mid \text{'I love movie'}) = \frac{P(\text{I love movie} \mid +)\, P(+)}{P(\text{I love movie})}$$

$P(+, \text{I love movie})$

$P(\text{I love movie})$

$$\propto P(\text{I} \mid +) \cdot P(\text{love} \mid +)\, P(\text{movie} \mid +)\, P(+)$$

$$P(- \mid \text{'I love movie}) \propto P(\text{I} \mid -) \cdot P(\text{love} \mid -)\, P(\text{movie} \mid -)\, P(-)$$

Priors

$$P(+) = \frac{3}{5}$$

$$P(-) = \frac{2}{5}$$

"I love movie": [1 1 1 0 0 0 0 0]

$P('\text{I}' \mid +) = $ ~~1/3~~ 1/7

$P('\text{love}' \mid +) = $ ~~1/3~~ 1/7

$P('\text{movie}' \mid +) = $ ~~2/3~~ 2/7

$P('\text{I}' \mid -) = $ ~~1/3~~ 1/5

$P(\text{love} \mid -) = $ ~~0~~ 0

$P(\text{movie} \mid -) = $ ~~1~~ 2/5

Any term with probability 0 will result in a 0 prob. for the entire class!

$P(+ \mid \text{'...'}) =$

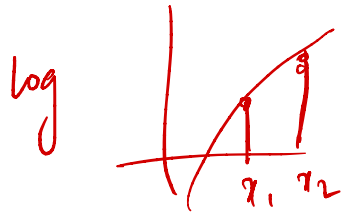
$P(- \mid \text{'...'}) = 0$

Laplacian smoothening

$$P(\text{word} \mid \text{class}) = \frac{\text{count of word in class} + 1}{\text{Total words in class} + \text{No. of unique words}}$$

Adding a row of 1's.

No change in priors.



$x_2 > x_1$

$\log x_2 > \log x_1$

Example

Movie Recommendation System

Suppose, we have a set of users, with preferences:

	M1 Top Gun Action	M2 Borat Comedy	M3 Whiplash Drama	M4 ... Action	M5 ... Comedy	M6 Romance
U1	1	0	?	?	1	?
U2	0	1	1	0	0	1
U3	1	1	0	1	1	0
U4	1	0	1	1	0	1
U5	0	1	0	0	1	0

Aim: Will user 1 like M3, M4 & M6?

Assumption: Preference of one genre is independent of preference of another?

Objective: $P(U1 = \text{like} \mid \text{Drama}) = ?$

$P(U1 = \text{like} \mid \text{Action}) = ?$

$P(U1 = \text{like} \mid \text{Romance}) = ?$

18-662 Homework 2

Feb 22, 2024
Due date: March 3, 2024

Problem 1: Maximum Flow for Image Segmentation [20 points]

In this problem, you will implement the Ford-Fulkerson algorithm to compute maximum flow. As part of this algorithm implementation, you will also have to implement an (uninformed) search algorithm, such as breadth-first search (BFS), depth-first search (DFS), or iterative-deepening search as a helper method.

Our goal in this problem is to perform image segmentation, which is the process of identifying and grouping pixels in an image based on which object they belong to. An example of this is shown in Figure 1, however, in your implementation, you will only be segmenting an image into two segments.

Figure 1. Image Segmentation Example

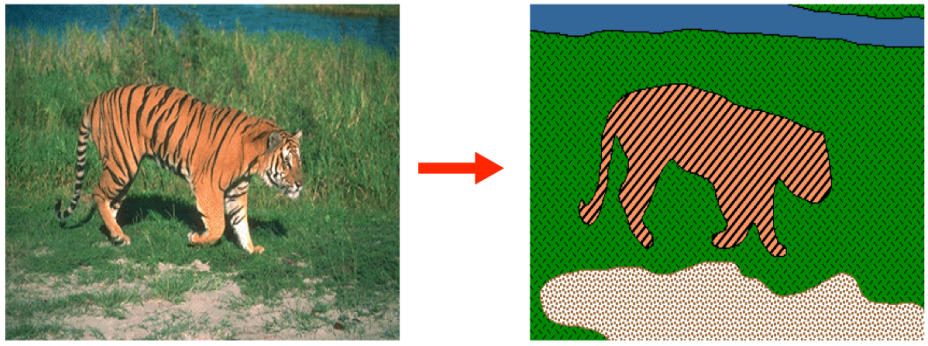


Image Credit: Stanford AI Lab
https://ai.stanford.edu/~syyeung/cvweb/tutorial3.html

Questions to answer before writing any code:

1. How does min-cut relate to this problem? Why is it true that the min-cut solution of a graph is equivalent to the max-flow solution? Draw examples if necessary.

2. Using your knowledge of min-cut, explain why solving the maximum flow problem is a valid method for image segmentation.

Requirements and Advice (read before implementing)

- Your code should include an implementation of the Ford-Fulkerson algorithm as well as an implementation of an uninformed search algorithm (as a helper to the Ford-Fulkerson algorithm).

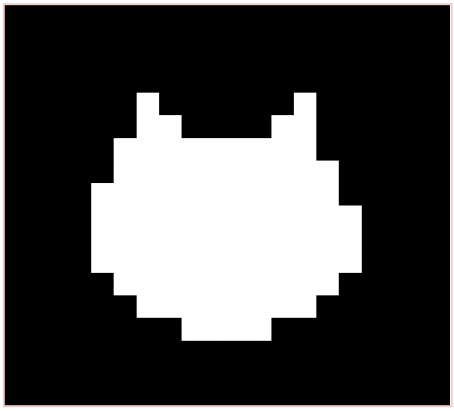
- You can assume that you will only be segmenting an image into **two** segments (never more and never less). This is to mirror the source and sink in the maximum flow problem.
- Your code should have some method of converting an image to a graph since the Ford-Fulkerson algorithm will require a graph structure.
    - The nodes of your graph should correspond to the pixels of the image. There should be as many nodes as pixels.
    - For every pair of nodes, there should be an edge connecting them if and only if they correspond to neighboring pixels in the image. The weight of the edge should be some measure of pixel similarity, and a greater edge weight should be assigned when the pixels are more similar. You can use the absolute difference of pixel values as edge weight (or another similarity measure you choose that meets the requirements).
- Develop your code on simple graphs (containing only a handful of nodes) and **not** images. This will help you debug as you go.
- You may have to play around with how you determine which pixels are the source and sink. The simplest method is setting your source as some pixel in a corner and the sink as some pixel in the center of the image (i.e. where the object is most likely to be).
- In addition to the implementation, you should submit a segmentation of cat.png. Figure 2. is an example of the desired segmentation.
- You should use the following code to load, view, and store the image as a Numpy array (it will be much simpler if you input binary images to your code):

```
from PIL import Image, ImageOps
img_path = cat.png # get this photo from Canvas. file name (or path if
image isn't in the same directory)
img = Image.open(img_path) # define img using PIL
img = ImageOps.grayscale(img) # make the image grayscale
img.show() # view image

img = np.array(img).astype(np.float32) # convert img to ndarray
img = (img < 0.5).astype(int) # binarize the image
print(img.shape)
h, w = img.shape # get height and width of the img
```

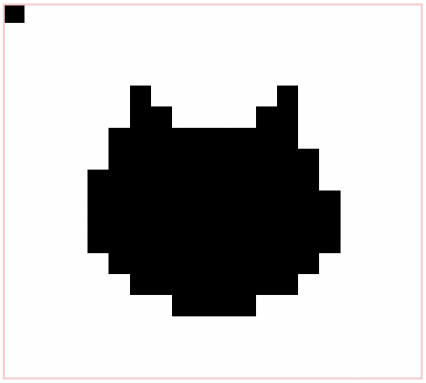
Figure 2. Desired Results

Input (low-resolution) cat.png

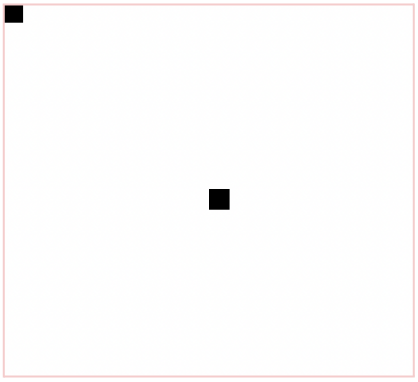


Binary input to algorithm

Resulting segmentation:



Nodes reachable from source



Nodes reachable from sink

Note: the resultant segmentations in your code may vary. It is important however that at least one map (either the nodes reachable from the source or the nodes reachable from the sink result in a valid segmentation as above). As an extension, you are welcome to repeat this problem but for non-binary inputs to the algorithm instead.

Questions to answer after implementation is complete:

1. Analyze the time complexity of the code you implemented. What could you change to reduce the runtime? (Note: no need to implement these changes—a few comments will suffice.)

2. What changes could you make to reduce the space (memory) requirement of your code? (Note: no need to implement these changes—a few comments will suffice.)

Question: Constraint Satisfaction for Sudoku Problem

Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which has a single solution.

Rules of Sudoku:

1. Each row must contain the digits 1-9 without repetition.
2. Each column must contain the digits 1-9 without repetition.
3. Each of the nine 3x3 subgrids must contain the digits 1-9 without repetition.

A Constraint Satisfaction Problem (CSP) is a mathematical question defined as a set of objects whose state must satisfy a number of constraints or limitations. Sudoku can be seen as a CSP where each cell in the grid is an object, and the constraints are defined by the rules of the game. CSPs are solved using a combination of search strategies and constraint satisfaction techniques.
Constraint satisfaction involves reducing the search space by applying the rules of the game (constraints) to eliminate impossible values for each cell. Techniques such as constraint propagation can be used to further reduce the number of possible values for each cell by examining the constraints imposed by the already filled cells.

Your task is to implement a Sudoku solver in Python that effectively combines constraint satisfaction and search algorithms for the following puzzles:

		6		1		5		
			3					
	9				8			
	7				3		4	
					4		9	3
				6		2		
7		2				1		
	4						8	
1				7				

1					7		9	
	3			2				8
		9	6			5		
		5	3			9		
	1			8				2
6					4			
3							1	
	4							7
		7				3		

Problem 3: MiniMax and Alpha-Beta pruning for Optimal Decisions

You may use this visualization to understand how alpha beta pruning works:
https://pascscha.ch/info2/abTreePractice/

***This visualization is not related to the problem stated in the homework. It is just provided to get an understanding**

Part 1: [10 points]

Finding the optimal next move in modified Tic-Tac-Toe

Minimax is a Kind of Backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. Alpha-Beta pruning is an optimization technique that can work on top of the Minimax algorithm to decrease computation complexity.

Your task in this section is to build an AI system that plays a perfect game. The agent will be playing a modified version of Tic-Tac-Toe where the size of the grid could be greater than or equal to 3X3. You must use both Minimax and Minimax with Alpha-Beta pruning to solve this problem. The input to the system will be the current state of the game in terms of a grid (list of list of str). The program must be able to produce the following outputs:

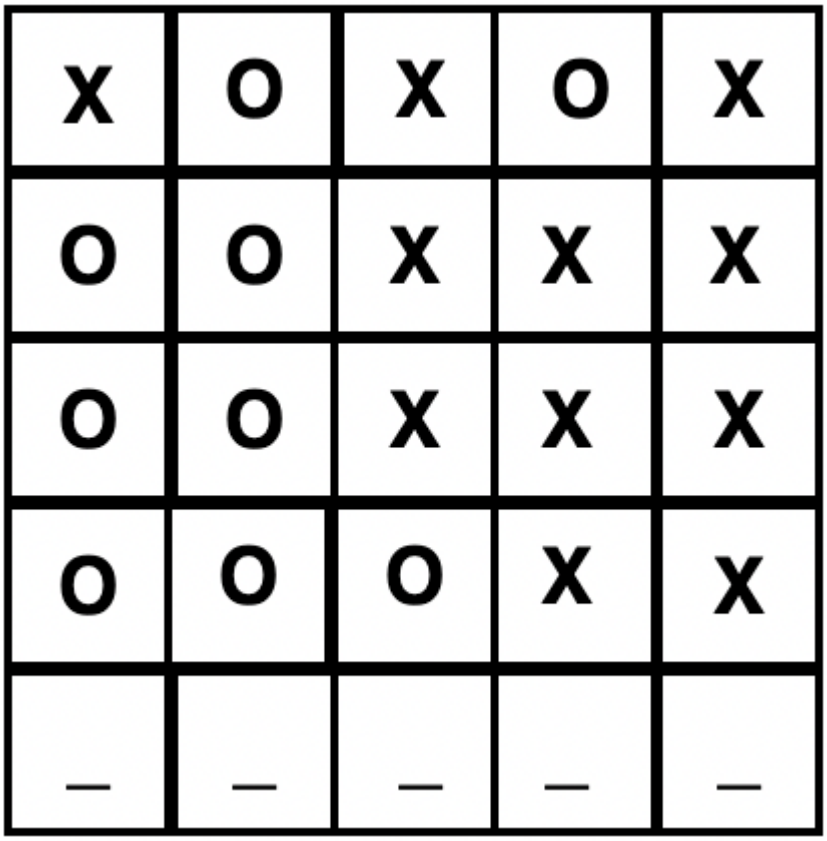
1. The Reward/value of the best move.
2. The position of the best move (the row and the column)
3. The time taken to get the optimal move in both cases (Minimax and Minimax with Alpha-Beta pruning)

Assumptions:

1. The grid will always be a square grid (the number of rows and columns will be equal)
2. 'x' will be the player and 'o' will be the opponent.
3. The player will win if he/she is able to draw a row/column/diagnol from one end of the grid to another(i.e., the maximum length it can be extended).
4. If there are 2 optimal positions, pick one with a lower column id
5. Reward must be +10 in case the player wins, -10 if the opponent wins and 0 if it is a tie. The priority must always be for the player to win.
6. The rows and columns are numbered from 0 to n.

Example:

Input:



board = [

['x', 'o', 'x', 'o', 'x'],

['o', 'o', 'x', 'x', 'x'],

['o', 'o', 'x', 'x', 'x'],

['o', 'o', 'o', 'x', 'x'],

['_', '_', '_', '_', '_']

]

Output:
The value of the best Move is : 10

The Optimal Move using minimax is :

ROW: 4 COL: 4

time taken by minimax: 0.0014388561248779297

The value of the best Move is : 10

The Optimal Move using alpha beta pruning is :

ROW: 4 COL: 4

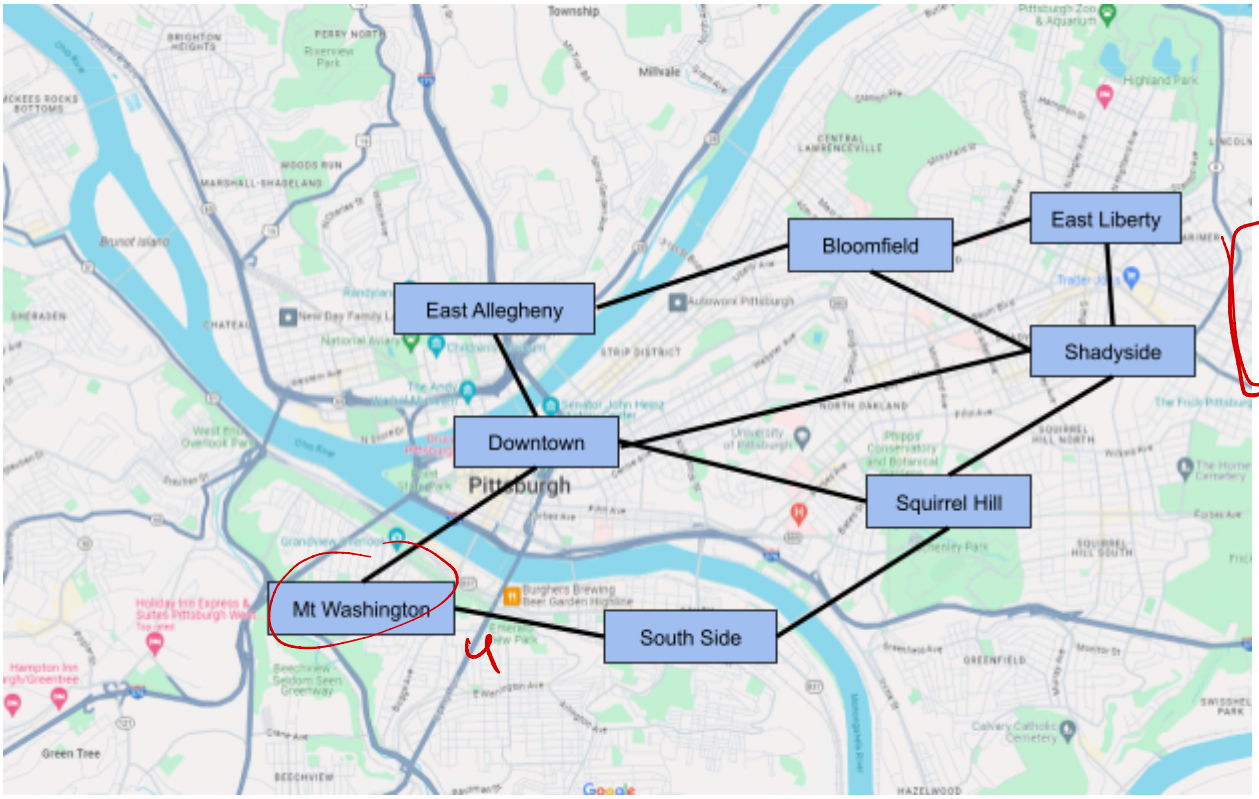
time taken by minimax with alpha beta pruning: 0.0009059906005859375

The above is just one test case, more test cases will be considered for grading this question.

Part 2 [5 points]

Can this be an Expectimax problem? Please provide a justification for your answer. Explain why one of them would work better than the other.

Evaluating the Public Transportation Network in Pittsburgh's Suburbs



The functionality of public transport systems within city confines plays a crucial role in well-being. Pittsburgh, with its distinctive geographical features and varied neighborhoods, poses significant challenges for the planning of public transportation. Acknowledging this, the Pittsburgh Regional Transit (PRT) has highlighted the necessity for an in-depth examination of the public transport network to improve service linkage and effectiveness, particularly for those suburbs that are currently under-served by existing transit routes.

As an analyst at PRT, your task is to pinpoint suburbs that are lacking in direct bus or train routes which enable travel to the city center in under 30 minutes. Moreover, the PRT aims to view this analysis through an equity lens, ensuring neighborhoods with larger populations of at-risk groups (for instance, the elderly or low-income families) are provided sufficient services. The objective is to suggest enhancements to the service that find a balance between efficiency and fairness.

You need to model Pittsburgh's public transportation framework using a weighted graph, where neighborhoods are represented as nodes and travel times as the weighted edges. For this, you need to design a weight metric that could consider both travel time, bus frequency, and equity factors, such as travel time * population or alpha * travel time + beta * population, and use this model to identify neighborhoods that are underserved by the public transit system. This should

take into account both the directness of the connections and the additional goals related to equity and service sufficiency.

Input Required:

1. A graph depicting the connections between Pittsburgh's neighborhoods.
2. Each edge should carry a weight denoting the travel time, frequency, and equity considerations. You should design your weight, such as travel time * population/frequency, alpha * travel time + beta * population… (Note: these are just illustrative, they might not be logical. You need to construct something logical, balancing the trade-offs.)

Output Expected:

1. A list of neighborhoods that are currently under-served by the public transit system.
2. A plan to enhance the service, which should cover suggestions for more direct connections and assurance that capacity meets the demand.

Example Scenario:

n = 10 (number of neighborhoods)
edges = [[1, 0, 15, 3, 200], [2, 0, 25, 1, 150], [3, 0, 22, 1, 300], ...]
Each edge is detailed as [src_node, dest_node, travel_time, bus_frequency (number of buses per hour), population].
{Use arbitrary numbers for the travel times, and assume numbers for the bus frequency (less than 6) and populations.}

Deliverables:

1. Construct the public transit network as a weighted graph.
2. Identify neighborhoods lacking connections under 30 minutes.
3. Factor in demographic data to address the needs of populations.
4. Suggest improvements that cater to demand.
5. Generate a comprehensive report that outlines your methodology, analysis, outcomes, and recommendations.

To submit:

1. The source code for the analysis and algorithm implementation.
2. A write-up that includes the methodology (including design of weight), findings, recommendations, and considerations for addressing the needs of vulnerable populations. How does the design of the weight affect your results?

Hint : Dijkstra

A* search and Heuristic: Solving a Maze

Implement the A* search algorithm to find the shortest path through a maze from a start point to a goal point. You will write a function that takes a maze represented as a list of lists, with walls, open spaces, a start point, and an end point, and returns the path from start to end as a list of coordinates. The maze you will solve is shown in the figure below.

We provided you with a python code that includes the required code structure. Your job is to fill out the function `a_star_search()` function that takes `maze, start, and goal` as inputs. We also provided you with the subfunctions `get_neighbors()` and `heuristic()` for you to use in observing the neighboring maze spots and calculating the Manhattan distance, respectively.

Important Note: We are giving you the maze only as a unit test. However, your code should solve any other maze on the same grid as well. Your solution should return `None` or the phrase "Not solution" if there is no solution.

The maze

Start			■			
	■				■	
	■			■		
		■	■			
	■			■		Goal

The solution (P is the path taken)

P	P	P	■	P	P	P
	■	P	P	P	■	P
	■			■		P
		■	■			P
	■			■		P

We expect to see this output from your code (# represents obstacle):

```
P P P # P P P
. # P P P # P
. # . . # . P
. . # # . . P
. # . . # . P
```

Problem 6: Two-Player Game [15 points]

In this problem we will extend the previous problem. Your task is to design an *adversarial* agent that will compete against you in N-puzzle by trying to place the tiles in the opposite order.

Desired Outcomes of the Game

8	**7**	**6**
5	**4**	**3**
2	**1**	**0**

The opponent's desired end state

1	**2**	**3**
4	**5**	**6**
7	**8**	**0**

The agent's desired end state

Game description:

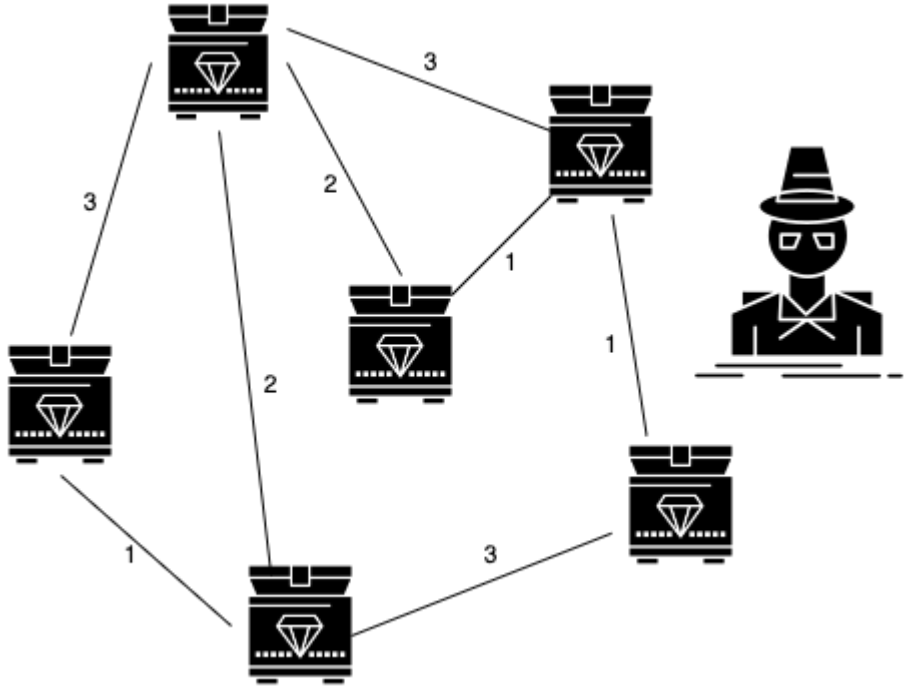
- This is a zero-sum game (i.e. the players take turns and the net gain and loss from each round is zero). You will define you own zero-sum reward system (i.e. when and how many points are assigned at each step).
- The problem will be simplest if you assume that the agent is trying to minimize a heuristic and that the opponent is trying to maxize the same heuristic or vice versa. You can choose any (appropriate) heuristic. This heuristic should be used to determine the value of each final position (i.e. the leaves of the game tree).
- The opponent is adversarial (not stochastic) and the opponent should view the agent as an adversarial (not stochastic) opponent.
- Define an `isEnd()` function that you use to determine whether the game is over.

Things to keep in mind:

- You can use either minimax or minimax with alpha-beta pruning to search the game tree.
- If you make any other assumptions or design choices, include an explanation along with your code submission.

Problem 7: Minimum Spanning Tree, Kruskal's Algorithm, Prim's Algorithm [10 points]

Suppose you are an archaeologist who is trying to plan a trip to visit a set of ancient ruins located in different parts of a country. You want to visit each of the ruins at least once, but you also want to minimize the total distance that you need to travel in order to see them all. Each ruin has its own unique features and historical significance, and you are excited to learn as much as you can about each one. You can visualize this scenario in the form of a graph like this:



The above graph is just for visualization, it is not a test case to consider.

The nodes are the ruins and the weight on the edges(bi-directional) correspond to the distance (in thousands of km) you need to travel in order to move from 1 ruin to the other. To solve this problem, you must find the optimal route that will allow you to visit each ruin while minimizing the total distance traveled.

Your program should take as input a list of lists defining the adjacency matrix of a graph. The edge weights of 2 connecting nodes must be stored at indices row 0 and col 1 i.e., if the distance between ruin 0 and ruin 1 is 3000 km, graph[0][1] = 3

example input:
graph = [[0, 9, 0, 1, 0],
[2, 0, 3, 7, 5],
[0, 3, 9, 0, 7],
[4, 8, 1, 0, 9],
[0, 3, 6, 9, 0]]

The program must be able to output the edges and the weights selected by the algorithm in the form (the order of the node in edges while printing does not matter i.e., 1 - 2 is equal to 2 - 1:

Edge	Weight
2 - 1	3
3 - 2	0
0 - 3	4
1 - 4	3

One more test case is provided in the jupyter notebook.