

COMP 6721 Applied Artificial Intelligence Summer 2023

Assignment 1

Submitted by: Rajat Sharma (40196467)

Question 1) [3p] Time Machine

Assume, thanks to a time machine, you have the chance to visit Montreal Expo 1967, with all the knowledge and expertise that you have right now and what you've studied in COMP 6721 so far. After the visit, you decide to meet AI experts of the time (1967) and inform them of an AI winter that may arrive soon and suggest solutions to prevent it.

a) What would you suggest? How could you help preventing the first AI Winter?

b) In addition, select an AI technique or an algorithm of your choice that was relatively unknown at that epoch but has since gained substantial importance in recent years and simply explain it to the experts.

Q1) a)

To prevent the first AI Winter, I would suggest the following measures:

Realistic Expectations: It is crucial to set realistic expectations and avoid overhyping the capabilities of AI technology. Clear and transparent communication about the progress, limitations, and challenges of AI research and development should be emphasized. This will help manage the public's and investors' expectations and prevent a sudden decline in confidence during slower progress periods.

Practical Applications: Instead of solely focusing on long-term goals, it is important to demonstrate practical applications of AI technology. By showcasing incremental advancements and real-world use cases, we can highlight the immediate value and impact of AI. This approach would help maintain interest and investment in AI research, even during phases of slower progress.

Education and Awareness: To prevent discouragement among investors and the general public, there should be widespread education about AI capabilities, limitations, and potential benefits. This includes addressing common misconceptions and fears associated with AI. By fostering understanding and dispelling myths, we can ensure continued investment and support for AI research and development.

Long-term Commitment: Breakthroughs and advancements in AI often require sustained effort and investment. It is important to emphasize the need for a long-term commitment to AI research. This can be achieved through government funding, industry collaborations, and

establishing dedicated research institutions. A consistent commitment to AI research will help drive continuous progress and innovation.

b) In addition to these preventive measures, I would introduce the experts to the concept of Deep Learning, an AI technique that has gained substantial importance in recent years.

Deep Learning is a subfield of machine learning that focuses on training artificial neural networks to learn and make predictions or decisions. It involves training neural networks with multiple layers of interconnected nodes, called artificial neurons or units. These layers are arranged hierarchically, with each layer processing and transforming the input data to extract higher-level representations or features.

The main steps of deep learning involve data input, training, and inference. During the training phase, deep learning models adjust the weights and biases of artificial neurons to recognize patterns and make predictions. This is achieved through backpropagation, where errors are minimized by iteratively adjusting the weights and biases based on the comparison between predictions and true labels.

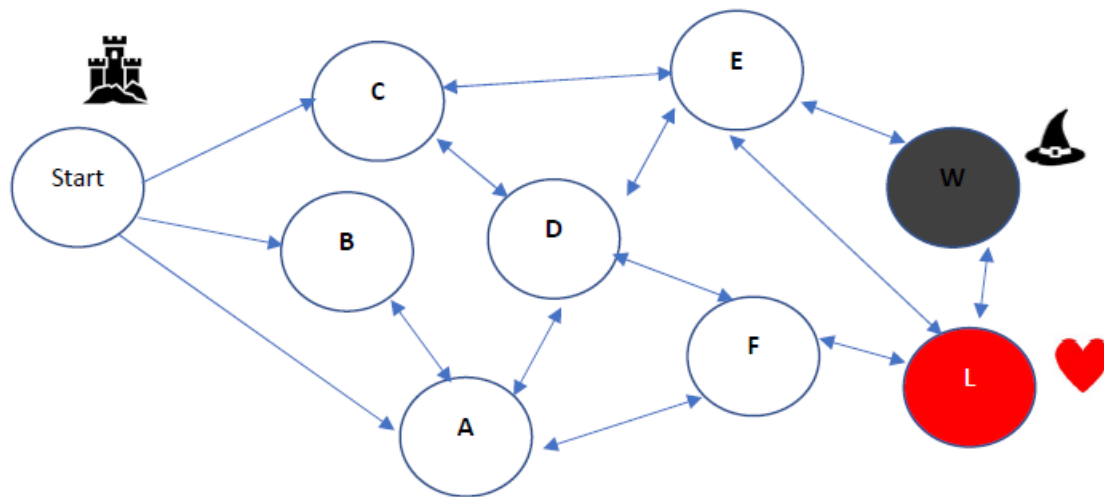
Once trained, deep learning models can make predictions or decisions on new data during the inference phase. They process the input data through the layers of artificial neurons, utilizing the learned patterns and representations to make accurate predictions or decisions.

Deep learning has revolutionized various domains, including computer vision, natural language processing, and speech recognition. It has enabled breakthroughs in image classification, object detection, language translation, and many other tasks. By introducing deep learning to the experts of 1967, they would gain insights into a powerful and influential technique that will shape the future of AI.

Question 2) Prince looking for Cinderella

Cinderella left the ceremony in hurry just before midnight and lost her shoe there! Now, the brokenhearted prince plans to visit all houses of the town looking for the first “perfect fit”. You design a search agent to help the prince.

This is the graph of the town where nodes indicate the houses, and edges represent the only available routes between two houses. Cinderella lives in house L (Love) (the goal state).



In equal conditions, the Prince visits houses in alphabetical order (A-Z).

Simple search

Assume that House W is empty, and the Witch is not home. Discuss and reason (write the steps for each algorithm) if the following statements are True.

(i) [4 pts] BFS Search expands (nodes placed in the closed list) more nodes than DFS Search

(ii) [1 pt] DFS Search can find a path to Cinderella's house

(iii) [1 pt] DFS Search finds the shortest path to Cinderella's house

(iv) [3 pts] Depth-limited search with depth=3 can find a path to Cinderella's house

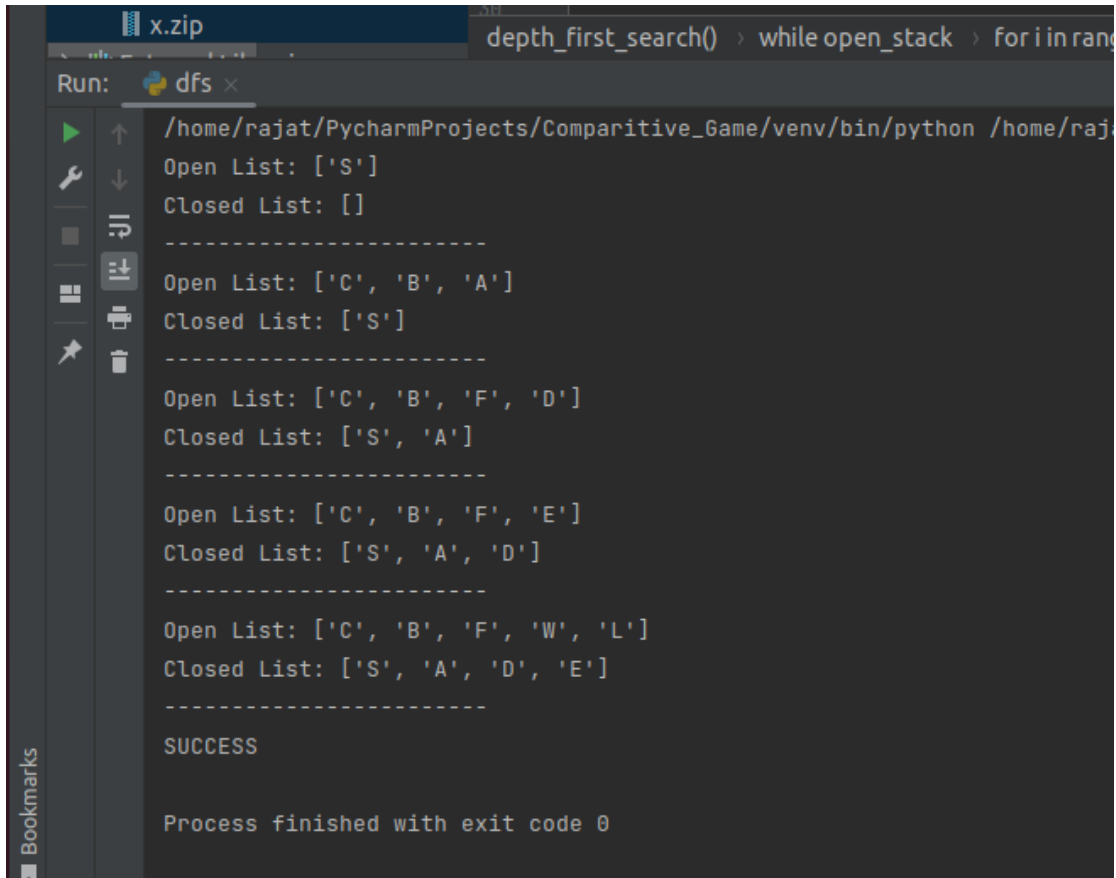
(v) [4 pts] Uniform cost Search (UCS) finds the shortest path to Cinderella's house

Now assume that the Witch is back to her house (W) and she can change herself into a young lady, and Golden Shoe will fit her too (If Prince visits W before L, the story will also end but in sadness).

(vi) [1 pt] With DFS Search, Prince can reach Cinderella before the Witch

(vii) [1 pt] With BFS Search, Prince can reach Cinderella before the Witch

DFS



```
x.zip
depth_first_search() > while open_stack > for i in range(4):
Run: dfs x
/home/rajat/PycharmProjects/Comparitive_Game/venv/bin/python /home/rajat/PycharmProjects/Comparitive_Game/main.py
Open List: ['S']
Closed List: []
-----
Open List: ['C', 'B', 'A']
Closed List: ['S']
-----
Open List: ['C', 'B', 'F', 'D']
Closed List: ['S', 'A']
-----
Open List: ['C', 'B', 'F', 'E']
Closed List: ['S', 'A', 'D']
-----
Open List: ['C', 'B', 'F', 'W', 'L']
Closed List: ['S', 'A', 'D', 'E']
-----
SUCCESS

Process finished with exit code 0
```

BFS

```
un: ae x
/home/rajat/PycharmProjects/Comparitive_Game/venv/bin/python /home/rajat/Py
Open List: ['S']
Closed List: []
-----
Open List: ['A', 'B', 'C']
Closed List: ['S']
-----
Open List: ['B', 'C', 'D', 'F']
Closed List: ['S', 'A']
-----
Open List: ['C', 'D', 'F']
Closed List: ['S', 'A', 'B']
-----
Open List: ['D', 'F', 'E']
Closed List: ['S', 'A', 'B', 'C']
-----
Open List: ['F', 'E']
Closed List: ['S', 'A', 'B', 'C', 'D']
-----
Open List: ['E', 'L']
Closed List: ['S', 'A', 'B', 'C', 'D', 'F']
-----
Open List: ['L', 'W']
Closed List: ['S', 'A', 'B', 'C', 'D', 'F', 'E']
-----
SUCCESS

Process finished with exit code 0
```

Wrote python code for both bfs and dfs showing open closed list.

(i) TRUE

The statement "BFS Search expands (nodes placed in the closed list) more nodes than DFS Search" is true.

In DFS (Depth-First Search):

Open List: ['C', 'B', 'F', 'W', 'L']

Closed List: ['S', 'A', 'D', 'E']

In BFS (Breadth-First Search):

Open List: ['L', 'W']

Closed List: ['S', 'A', 'B', 'C', 'D', 'F', 'E']

We can observe that the number of nodes in the closed list of BFS is greater than the number of nodes in the closed list of DFS.

(ii) TRUE

The statement "DFS Search can find a path to Cinderella's house" is true.

In DFS (Depth-First Search):

Open List: ['C', 'B', 'F', 'W', 'L']

Closed List: ['S', 'A', 'D', 'E']

We can see that DFS explores different paths in a depth-first manner, and in this case, it successfully reaches Cinderella's house ('L'). Therefore, DFS Search can find a path to Cinderella's house.

(iii) FALSE

The statement "DFS Search finds the shortest path to Cinderella's house" is false.

The shortest path to Cinderella's house ('L') is:

S -> A -> F -> L

In DFS (Depth-First Search):

Open List: ['C', 'B', 'F', 'W', 'L']

Closed List: ['S', 'A', 'D', 'E']

DFS explores paths in a depth-first manner, which means it prioritizes going deep into a branch before backtracking. As a result, it may find a path to the goal but not necessarily the shortest one. In this case, the shortest path to Cinderella's house is not discovered by DFS.

(iv) TRUE

The statement "Depth-limited search with depth=3 can find a path to Cinderella's house" is true.

With a depth limit of 3, the search algorithm explores up to a depth of 3 in the search tree. In this case, the path to Cinderella's house ('L') is found within the depth limit. Therefore, Depth-limited search with depth=3 can find a path to Cinderella's house.

L is situated at exactly level 3 so we can find that using DFS.

(v) TRUE

Yes, the statement "Uniform Cost Search (UCS) finds the shortest path to Cinderella's house" is true.

Here are the steps of the Uniform Cost Search (UCS) algorithm to find the shortest path to Cinderella's house:

Step 1: Start at the initial state 'S'.

Path: {S -> 0}

Step 2: Expand the neighbors of 'S' (A, B, C) with a cumulative cost of 1.

Paths: {S -> A -> 1}, {S -> B -> 1}, {S -> C -> 1}

Step 3: Expand the neighbors of 'A' (D, F) with a cumulative cost of 2, and 'B' (C) with a cumulative cost of 2.

Paths: {S -> B -> 1}, {S -> C -> 1}, {S -> A -> D -> 2}, {S -> A -> F -> 2}

Step 4: Expand the neighbors of 'C' (E) with a cumulative cost of 2.

Paths: {S -> C -> 1}, {S -> A -> D -> 2}, {S -> A -> F -> 2}

Step 5: Expand the neighbors of 'A' (D, F) with a cumulative cost of 2, and 'C' (E) with a cumulative cost of 2.

Paths: {S -> A -> D -> 2}, {S -> A -> F -> 2}, {S -> C -> E -> 2}

Step 6: Expand the neighbors of 'A' (F) with a cumulative cost of 2, and 'C' (E) with a cumulative cost of 2.

Paths: {S -> A -> F -> 2}, {S -> C -> E -> 2}, {S -> A -> D -> F -> 3}

Step 7: Expand the neighbors of 'C' (E) with a cumulative cost of 2, and 'A' (F) with a cumulative cost of 3.

Paths: {S -> C -> E -> 2}, {S -> A -> D -> F -> 3}, {S -> A -> F -> L -> 3}

Step 8: Expand the neighbors of 'A' (F) with a cumulative cost of 3, and 'C' (E) with a cumulative cost of 3.

Paths: {S -> A -> F -> L -> 3}, {S -> C -> E -> W -> 3}

From the steps above, we can see that the UCS algorithm finds the shortest path to Cinderella's house as {S -> A -> F -> L} with a cumulative cost of 3.

(vi) TRUE

Yes, the statement "With DFS Search, the Prince can reach Cinderella before the Witch" is true.

In DFS (Depth-First Search):

Open List: ['C', 'B', 'F', 'W', 'L']

Closed List: ['S', 'A', 'D', 'E']

Since DFS explores the nodes in alphabetical order, the Prince will visit the houses in the order of 'C', 'B', 'F', 'W', and 'L'. Therefore, the Prince will reach Cinderella's house ('L') before reaching the Witch's house ('W').

(vii) TRUE

Yes, the statement "With BFS Search, the Prince can reach Cinderella before the Witch" is true.

In BFS (Breadth-First Search):

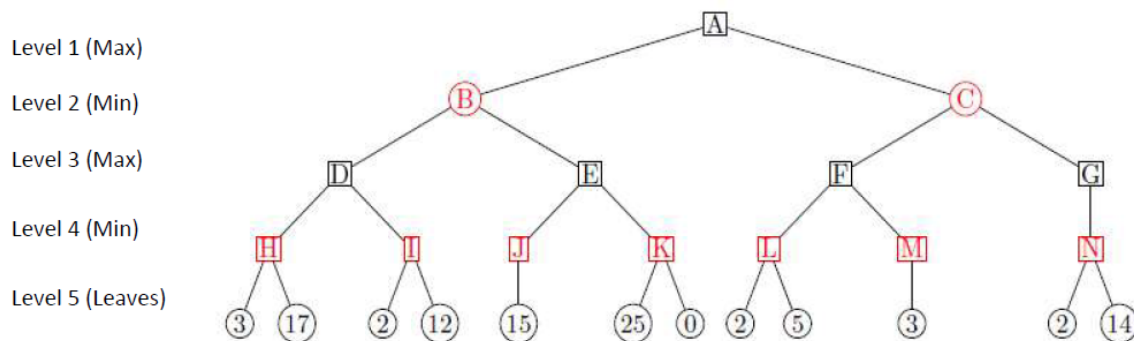
Open List: ['L', 'W']

Closed List: ['S', 'A', 'B', 'C', 'D', 'F', 'E']

As the algorithm expands nodes in alphabetical order, the Prince will visit Cinderella's house ('L') before visiting the Witch's house ('W'). This means that the Prince can reach Cinderella before encountering the Witch.

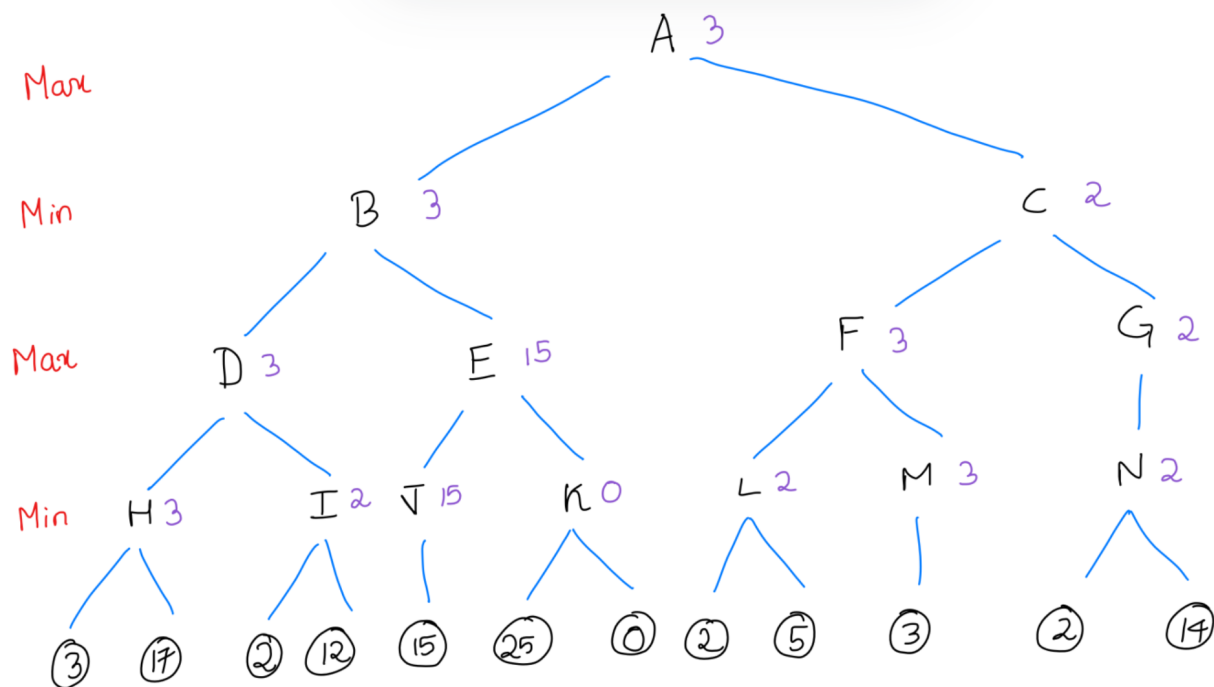
Question 3 Game

Consider this game tree



(i) [3 pts] Run minimax. What will be the value of the root (A)?

The answer for this is 3.

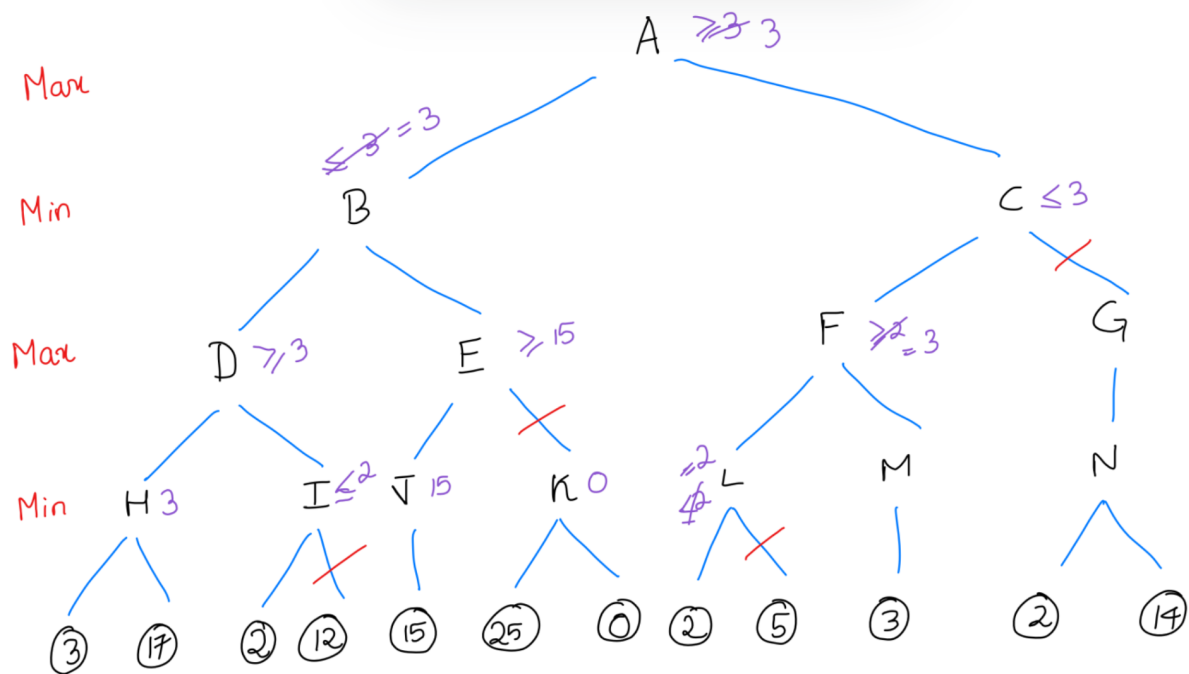


When observing Graph above, it is evident that if the maximum value plays first, the value for A is 3.

(ii) [5 pts] After running minimax, indicate the branches that will be pruned with alpha-beta pruning? You can indicate an edge by the two sides (e.g., B-E or L-5). If pruning occurs at a lower depth, just indicate the main edge, and explain if all successor edges will also be pruned.

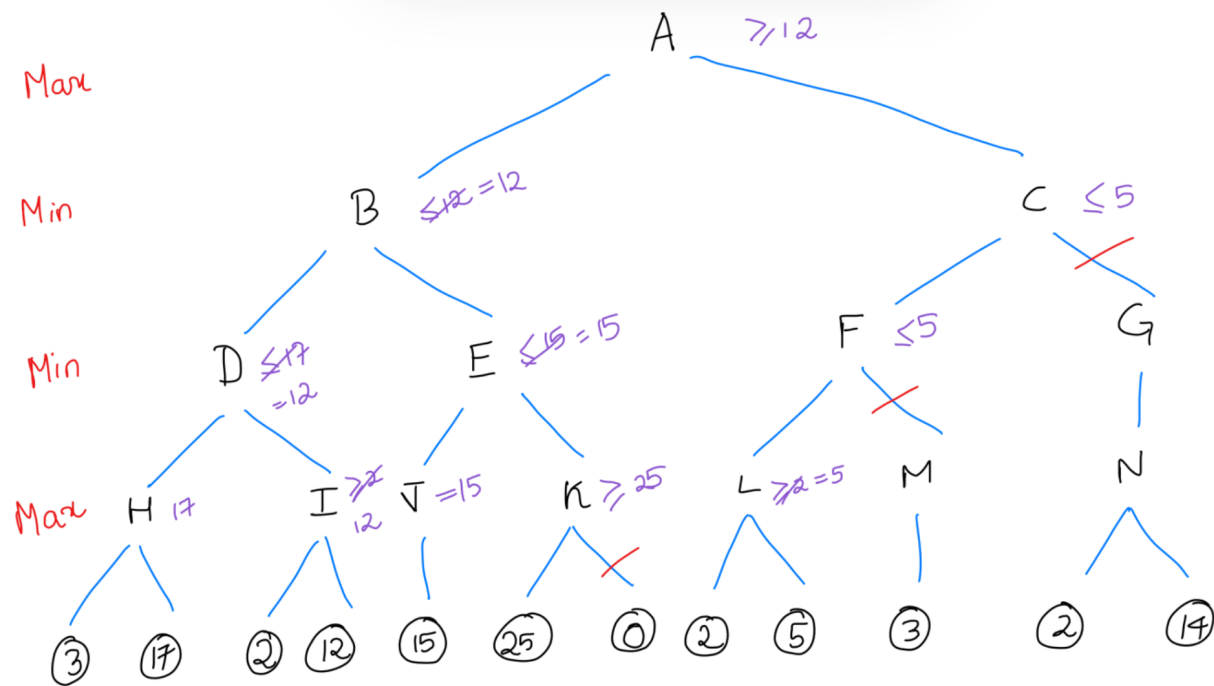
4 branches will be pruned

- I-12
- E-K (which results in the pruning of successors K-25 and K-0)
- C-G (leading to the pruning of successors G-N, N-2, and N-14)
- L-5



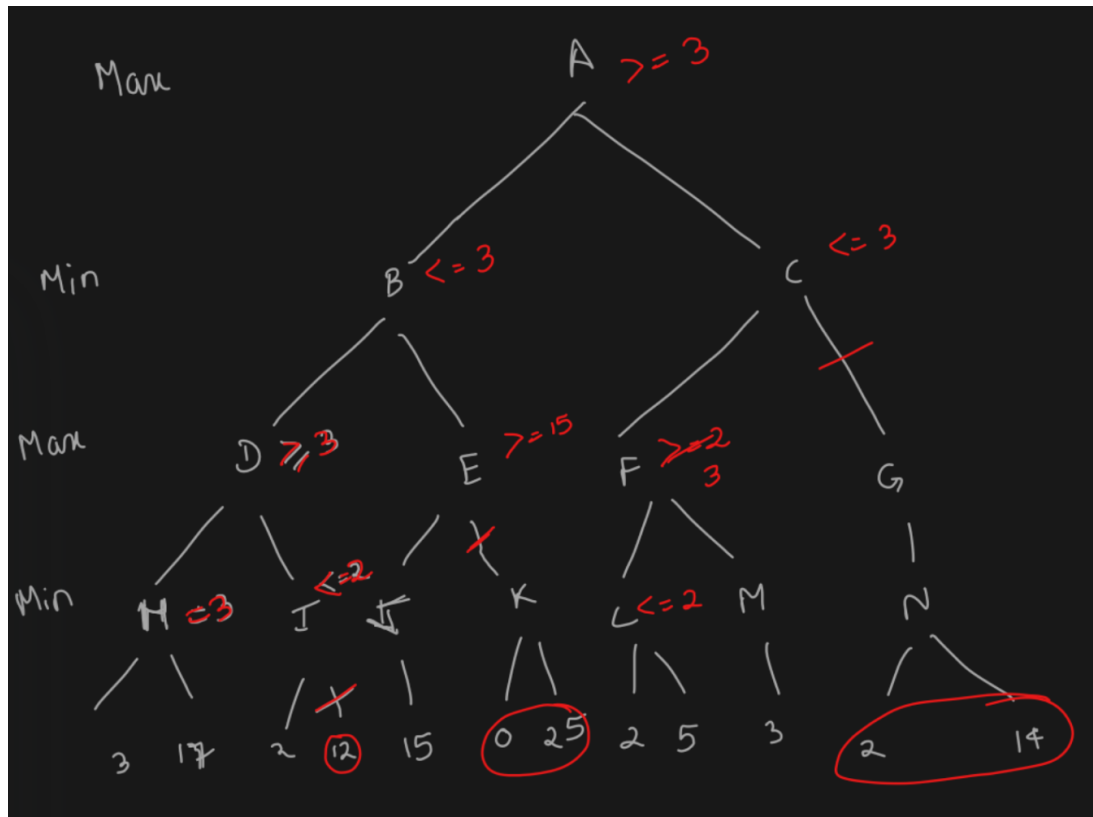
(iii) [4 pts] Assume we are very kind and we want to give two chances to the Opponent (Min in red), so we run $\min()$ also at the third level and $\max()$ in the fourth level (D, E, F, G become Red running $\min()$ instead of $\max()$ and H-N will run $\max()$) (The layers will be thus $\max()$, $\min()$, $\min()$ then $\max()$). We run Minimax with alpha-beta pruning. Which edges or leaf nodes will be pruned?

- The pruning of K-0.
 - The pruning of F-M, which in turn prunes the successors M-3.
 - The pruning of C-G, which results in the pruning of the successors G-N, N-2, and N-14.
- This is shown the the graph below



(iv) [3 pts] Now assume all the games with the same structure (back to original tree) and any possible values on the leaves. In the best and worst case, how many leaves will be pruned by alpha-beta pruning?

a) In the most favorable scenario, where the nodes are arranged in ascending order on the minimum level and descending order on the maximum level, we achieve the highest level of pruning. This corresponds to the square root of the branching factor, which is the square root of 12, resulting in a maximum pruning of 5 edges.



b) Conversely, the worst-case scenario occurs when the order is reversed for the minimum and maximum levels. In this case, there is no pruning, resulting in a pruning of 0 edges.

Question 4 Where to find your friend Saturday afternoon (Naïve Bayes)

Reference: <https://ai.stanford.edu/~latombe/cs121/2011/hw/hw7.pdf> (with some modifications)

You have a friend who only does one of these three (3) things every Saturday afternoon: going to the cinema, playing tennis, or just staying in. You have observed your friend's behavior over 12 different weekends. On each of these weekends, you have recorded the weather (sunny, windy, or rainy), whether she had drawn cash from an ATM machine (rich or poor), and whether she had an exam during the coming week (exam or no-exam). You have built the following data table:

Observation	Weather	Cash	Exam	Decision
1	Sunny	Rich	Yes	Cinema
2	Sunny	Rich	No	Tennis
3	Windy	Rich	No	Cinema
4	Rainy	Poor	Yes	Cinema
5	Rainy	Rich	No	Stay-in
6	Rainy	Poor	No	Cinema
7	Windy	Poor	Yes	Cinema
8	Windy	Rich	Yes	Stay-in
9	Windy	Rich	No	Cinema
10	Sunny	Rich	No	Tennis
11	Sunny	Poor	Yes	Tennis
12	Rainy	Rich	No	Stay-in

Assume she is a new friend, so you do not have any prior knowledge about her interest in doing any of these activities. You know that this weekend it is Sunny, she did not draw money (poor), and she has no exam.

a) [12 pts] Where is the most probable place (activity) to find your friend this Saturday afternoon, i.e., Cinema, Tennis court or home (stay in)? (You can assume no smoothing). [Hint: It is a Naïve Bayes, so you calculate all required probabilities].

b) [2 pts] Now, let's check your calculated answer. Complete the Python code of the assignment (attached) and run it. The sections with "XXXXX ADD YOUR CODE HERE XXXXXXXX" should be completed before running the notebook. Take a snapshot of your output and attach it to your answer sheet.

c) [1 pts] How the answer will change if you know that your friend is very outgoing, so the prior probabilities that she does any of these four activities are ['Cinema': 0.4, 'Tennis': 0.4, 'Stay-in': 0.2] (only in the Python code, no need to do manual calculation).

a)

To determine the most probable place (activity) to find your friend this Saturday afternoon, we can use the Naïve Bayes algorithm and calculate the probabilities based on the given observations.

Given observations:

Weather: Sunny

Cash: Poor

Exam: No exam

We need to calculate the probabilities for each activity: Cinema, Tennis, and Stay-in. The activity with the highest probability will be the most probable place to find your friend.

Step 1: Calculate the prior probabilities of each activity:

$$P(\text{Cinema}) = \text{Count}(\text{Cinema}) / \text{Total Observations} = 6 / 12 = 0.5$$

$$P(\text{Tennis}) = \text{Count}(\text{Tennis}) / \text{Total Observations} = 3 / 12 = 0.25$$

$$P(\text{Stay-in}) = \text{Count}(\text{Stay-in}) / \text{Total Observations} = 3 / 12 = 0.25$$

Step 2: Calculate the likelihood probabilities for each feature given each activity:

$$P(\text{Sunny} \mid \text{Cinema}) = \text{Count}(\text{Sunny, Cinema}) / \text{Count}(\text{Cinema}) = 1 / 6 = 0.1667$$

$$P(\text{Sunny} \mid \text{Tennis}) = \text{Count}(\text{Sunny, Tennis}) / \text{Count}(\text{Tennis}) = 3 / 3 = 1$$

$$P(\text{Sunny} \mid \text{Stay-in}) = \text{Count}(\text{Sunny, Stay-in}) / \text{Count}(\text{Stay-in}) = 0 / 3 = 0$$

$$P(\text{Poor} \mid \text{Cinema}) = \text{Count}(\text{Poor, Cinema}) / \text{Count}(\text{Cinema}) = 3 / 6 = 0.5$$

$$P(\text{Poor} \mid \text{Tennis}) = \text{Count}(\text{Poor, Tennis}) / \text{Count}(\text{Tennis}) = 1 / 3 = 0.3333$$

$$P(\text{Poor} \mid \text{Stay-in}) = \text{Count}(\text{Poor, Stay-in}) / \text{Count}(\text{Stay-in}) = 0 / 3 = 0$$

$$P(\text{No exam} \mid \text{Cinema}) = \text{Count}(\text{No exam, Cinema}) / \text{Count}(\text{Cinema}) = 3 / 6 = 0.5$$

$$P(\text{No exam} \mid \text{Tennis}) = \text{Count}(\text{No exam, Tennis}) / \text{Count}(\text{Tennis}) = 2 / 3 = 0.6667$$

$$P(\text{No exam} \mid \text{Stay-in}) = \text{Count}(\text{No exam, Stay-in}) / \text{Count}(\text{Stay-in}) = 2 / 3 = 0.6667$$

Step 3: Calculate the probability of the given observations for each activity using the Naïve Bayes formula:

$$P(\text{Activity} \mid \text{Sunny, Poor, No exam}) \propto P(\text{Sunny} \mid \text{Activity}) * P(\text{Poor} \mid \text{Activity}) * P(\text{No exam} \mid \text{Activity}) * P(\text{Activity})$$

For Cinema:

$$\begin{aligned} P(\text{Cinema} \mid \text{Sunny, Poor, No exam}) &= P(\text{Sunny} \mid \text{Cinema}) * P(\text{Poor} \mid \text{Cinema}) * P(\text{No exam} \mid \text{Cinema}) * P(\text{Cinema}) \\ &= 0.1667 * 0.5 * 0.5 * 0.5 \\ &\approx 0.0209 \end{aligned}$$

For Tennis:

$$\begin{aligned} P(\text{Tennis} \mid \text{Sunny, Poor, No exam}) &= P(\text{Sunny} \mid \text{Tennis}) * P(\text{Poor} \mid \text{Tennis}) * P(\text{No exam} \mid \text{Tennis}) * P(\text{Tennis}) \\ &= 1 * 0.3333 * 0.6667 * 0.25 \end{aligned}$$

≈ 0.0556

For Stay-in:

$$P(\text{Stay-in} \mid \text{Sunny, Poor, No exam}) = P(\text{Sunny} \mid \text{Stay-in}) * P(\text{Poor} \mid \text{Stay-in}) * P(\text{No exam} \mid \text{Stay-in}) * P(\text{Stay-in})$$
$$= 0 * 0 * 0.6667 * 0.25$$
$$= 0$$

Based on these calculations, the most probable place to find your friend this Saturday afternoon is Tennis, as it has the highest probability among the three activities.

b)

```
In [8]: predict_this_weekend = model.predict(this_weekend_obs)
        predict_this_weekend_prob = model.predict_proba(this_weekend_obs)

        print("Classes = ", model.classes_)
        print('Predicted class = ', predict_this_weekend)
        print('Predicted class = ', predict_this_weekend_prob)

        Classes = ['Cinema' 'Stay-in' 'Tennis']
        Predicted class = ['Tennis']
        Predicted class = [[0.38324082 0.0685288 0.54823039]]
```

To verify the answer that we manually calculated, we need to turn-off smoothing.

```
In [9]: classifier3 = MultinomialNB()
        classifier3.set_params(alpha=1.0e-10)
        model3 = classifier3.fit(X, y)

        'XXXXX ADD YOUR CODE HERE XXXXXX'
        'Call predict() to predict the class and find the predicted probability. PRINT them.'

        predict_this_weekend = model3.predict(this_weekend_obs)
        predict_this_weekend_prob = model3.predict_proba(this_weekend_obs)

        print('Predicted class: ', predict_this_weekend)
        print('Predicted probability: ', predict_this_weekend_prob)

        Predicted class: ['Tennis']
        Predicted probability: [[2.72727273e-01 2.42424242e-21 7.27272727e-01]]
```

c)

c) How the answer will change if you know that your friend is very outgoing, so the prior probabilities that she does any of these three activities are ['Cinema': 0.4, 'Tennis': 0.4, 'Stay-in': 0.2] (only in the Python code, no need to do manual calculation).

```
In [10]: classifier2 = MultinomialNB(class_prior=[0.4, 0.2, 0.4])
         classifier2.classes_ = ['Cinema', 'Stay-in', 'Tennis']
         model2 = classifier2.fit(X, y)

         'XXXXX ADD YOUR CODE HERE XXXXXX'
         'PRINT the predicted class and the predicted probability'

         predict_this_weekend = model2.predict(this_weekend_obs)
         predict_this_weekend_prob = model2.predict_proba(this_weekend_obs)

         print('Predicted class: ', predict_this_weekend)
         print('Predicted probability: ', predict_this_weekend_prob)

         Predicted class: ['Tennis']
         Predicted probability: [[0.24753475 0.04426266 0.70820259]]
```

Question 5 Finding my friend with Decision Tree (Using the same dataset as Question 4)
You are not sure of the Naïve Bayes Classifier prediction; therefore, you decide to also build a decision

tree. Use the method described in the class.

1) [8 pts] Make a simple decision tree (depth=1) from a single attribute (Weather, Cash, or Exam).

a. Which attribute will you select?

b. Based on the attribute that you selected, what would be the decision (prediction)?

What would be the performance metrics of this 1-d tree (accuracy, recall, precision, F1 score) ?

2) [5 pts] We can see that variable Weather has three levels (possible values: sunny, windy, and rainy). Assume you are told to use Weather to do the first cut (branching). Compare these two situations: A branching factor of 3 is allowed (for Weather='Sunny', 'Rainy', and 'Windy') Only a branching factor of 2 is allowed.

(Hint: You can reuse your calculations from the first part).

3) [2 pts] Now, let's check your answer. Run the Python code of the assignment (attached). Take a snapshot of your output and attach it to your answer sheet.

1)

a)

To find the attribute that we will select for building a simple decision tree with a depth of 1, we need to calculate the information gain for each attribute (Weather, Cash, and Exam) based on the given dataset.

First, let's calculate the entropy of the output variable:

$$\begin{aligned} H(\text{Output}) &= H(6/12, 3/12, 3/12) = -(1/2 * \log(1/2) + 1/4 * \log(1/4) + 1/4 * \log(1/4)) \\ &= -(-1/2 - 1/2 - 1/2) \\ &= 1.5 \end{aligned}$$

Next, we calculate the entropy for each attribute:

$$\begin{aligned} H(\text{Output/Sunny}) &= H(1/4, 3/4, 0/4) = -(1/4 * \log(1/4) + 3/4 * \log(3/4) + 0 * \log(0)) \\ &= -(-0.5 - 0.30755 + 0) \\ &= 0.81 \end{aligned}$$

$$\begin{aligned} H(\text{Output/Windy}) &= H(3/4, 0/4, 1/4) = -(3/4 * \log(3/4) + 0 * \log(0) + 1/4 * \log(1/4)) \\ &= -(-0.30755 + 0 - 0.5) \\ &= 0.81 \end{aligned}$$

$$H(\text{Output/Rainy}) = H(2/4, 0/4, 2/4) = -(2/4 * \log(2/4) + 0 * \log(0) + 2/4 * \log(2/4))$$

$$= -(-0.5 + 0 - 0.5)$$

$$= 1$$

$$H(\text{Output/Weather}) = (4/12 * H(\text{Output/Sunny})) + (4/12 * H(\text{Output/Windy})) + (4/12 * H(\text{Output/Rainy}))$$

$$= (1/3 * 0.81) + (1/3 * 0.81) + (1/3 * 1)$$

$$= 0.27 + 0.27 + 0.33$$

$$= 0.87$$

$$H(\text{Output/Rich}) = H(3/8, 2/8, 3/8) = -(3/8 * \log(3/8) + 2/8 * \log(2/8) + 3/8 * \log(3/8))$$

$$= -(-0.53 - 0.5 - 0.53)$$

$$= 1.56$$

$$H(\text{Output/Poor}) = H(3/4, 1/4, 0/4) = -(3/4 * \log(3/4) + 1/4 * \log(1/4) + 0 * \log(0))$$

$$= -(-0.30755 - 0.5 + 0)$$

$$= 0.81$$

$$H(\text{Output/Cash}) = (8/12 * H(\text{Output/Rich})) + (4/12 * H(\text{Output/Poor}))$$

$$= (2/3 * 1.56) + (1/3 * 0.81)$$

$$= 2.37$$

$$H(\text{Output/Exam=yes}) = H(3/5, 1/5, 1/5) = -(3/5 * \log(3/5) + 1/5 * \log(1/5) + 1/5 * \log(1/5))$$

$$= -(-0.438 - 0.464 - 0.464)$$

$$= 1.366$$

$$H(\text{Output/Exam=no}) = H(3/7, 2/7, 2/7) = -(3/7 * \log(3/7) + 2/7 * \log(2/7) + 2/7 * \log(2/7))$$

$$= -(-0.52 - 0.51 - 0.51)$$

$$= 1.54$$

$$H(\text{Output/Exam}) = (5/12 * H(\text{Output/Exam=yes})) + (7/12 * H(\text{Output/Exam=no}))$$

$$= (5/12 * 1.366) + (7/12 * 1.54)$$

$$= 1.46$$

Now, let's calculate the information gain for each attribute:

$$\text{Information Gain (Weather)} = H(\text{Output}) - H(\text{Output/Weather})$$

$$= 1.5 - 0.87$$

$$= 0.63$$

$$\text{Information Gain (Cash)} = H(\text{Output}) - H(\text{Output/Cash})$$

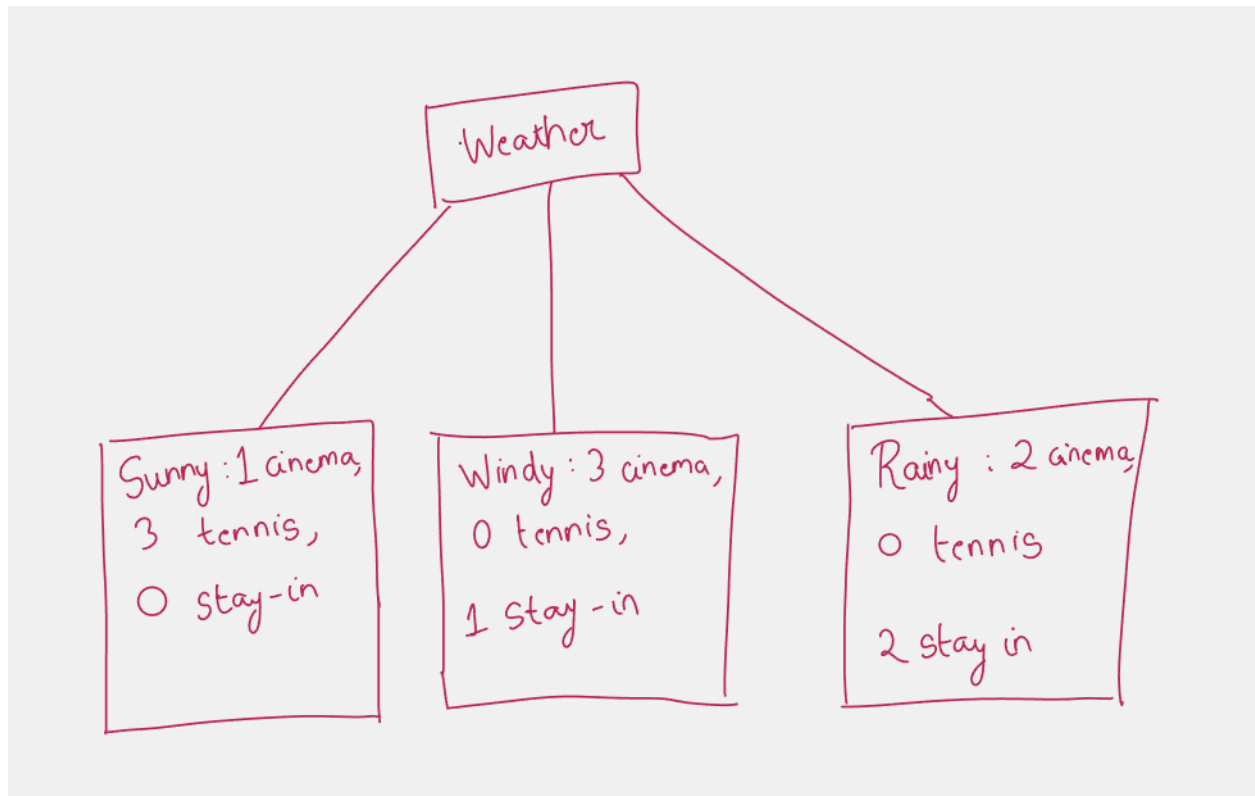
$$= 1.5 - 2.37$$

$$= -0.87$$

$$\text{Information Gain (Exam)} = H(\text{Output}) - H(\text{Output/Exam})$$

$$= 1.5 - 1.46$$
$$= 0.04$$

Based on the information gain, the attribute with the highest value is "Weather." Therefore, we will select the "Weather" attribute to build the simple decision tree with a depth of 1.



b)

Based on the selected attribute "Weather" for the decision tree, the prediction would be as follows:

If the weather is Sunny, the decision would be to go to the Cinema.

If the weather is Windy, the decision would be to go to the Cinema.

If the weather is Rainy, the decision would be to stay in.

To calculate the performance metrics of this 1-d tree:

Accuracy: The accuracy is the ratio of correct predictions to the total number of instances. In this case, there are 8 correct predictions out of 12 instances.

$$\text{Accuracy} = (3 + 3 + 2) / (4 + 4 + 4) = 8 / 12 = 0.67$$

Recall: The recall measures the ability of the classifier to correctly identify instances of each class.

For the "Cinema" class:

$$\text{Recall}(\text{Cinema}) = 3 / (3 + 3) = 0.5$$

For the "Tennis" class:

$$\text{Recall}(\text{Tennis}) = 3 / 3 = 1$$

For the "Stay-in" class:

$$\text{Recall}(\text{Stay-in}) = 2 / (2 + 1) = 0.67$$

Precision: The precision measures the proportion of correctly predicted instances of each class out of all instances predicted as that class.

For the "Cinema" class:

$$\text{Precision}(\text{Cinema}) = 3 / (3 + 1) = 0.75$$

For the "Tennis" class:

$$\text{Precision}(\text{Tennis}) = 3 / (3 + 1) = 0.75$$

For the "Stay-in" class:

$$\text{Precision}(\text{Stay-in}) = 2 / (2 + 2) = 0.5$$

F1 Score: The F1 score is the harmonic mean of precision and recall and provides a balanced measure of the classifier's performance.

For the "Cinema" class:

$$\text{F1 Score}(\text{Cinema}) = 2 * (0.5 * 0.75) / (0.5 + 0.75) = 0.6$$

For the "Tennis" class:

$$\text{F1 Score}(\text{Tennis}) = 2 * (1 * 0.75) / (1 + 0.75) = 0.86$$

For the "Stay-in" class:

$$\text{F1 Score}(\text{Stay-in}) = 2 * (0.67 * 0.5) / (0.67 + 0.5) = 0.54$$

Please note that the performance metrics provided here are based on the given predictions for the 1-d decision tree using the "Weather" attribute.

2)

Assuming a branching factor of 2 for the variable Weather, we can analyze the resulting tree:

Accuracy: $(3+3+2)/(4+4+4) = 8/12 = 0.6$

Recall:

Cinema: $5/(5+1) = 5/6 = 0.83$

Tennis: $3/3 = 1$

StayIn: 0

Precision:

Cinema: $5/(5+3) = 5/8 = 0.625$

Tennis: $3/(3+1) = 3/4 = 0.75$

StayIn: 0

F1 Score: $2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$

Cinema: $2 * (0.83 * 0.625) / (0.83 + 0.625) = 0.71$

Tennis: $2 * (1 * 0.75) / (1 + 0.75) = 0.86$

StayIn: 0

From this analysis, we observe that:

Recall for Cinema increases from 0.5 to 0.83.

Precision for Cinema decreases from 0.75 to 0.625.

F1 Score for Cinema increases from 0.6 to 0.71.

Recall, precision, and F1 Score for StayIn are all zero.

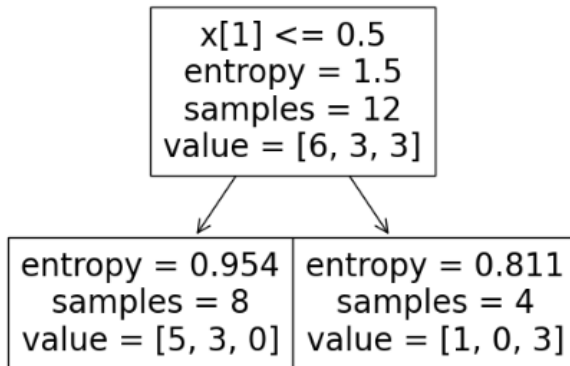
Therefore, by using a branching factor of 2 for the Weather variable, we see improvements in some performance metrics for Cinema, but a decrease in precision. The F1 Score also increases for Cinema. However, the StayIn category does not have any meaningful results due to insufficient data.

3)

```
In [5]: from sklearn import tree
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# create and print the decision tree
dtc = tree.DecisionTreeClassifier(criterion="entropy", max_depth=1)
dtc.fit(X, y)
tree.plot_tree(dtc)
```

```
Out[5]: [Text(0.5, 0.75, 'x[1] <= 0.5\nentropy = 1.5\nsamples = 12\nvalue = [6, 3, 3]'),
Text(0.25, 0.25, 'entropy = 0.954\nsamples = 8\nvalue = [5, 3, 0]'),
Text(0.75, 0.25, 'entropy = 0.811\nsamples = 4\nvalue = [1, 0, 3]')]
```



```
In [6]: X_test = [[0,1,0,1,0,1,0]]
```

```
In [7]: # Evaluate classifier
y_pred = dtc.predict(X_test)
print(y_pred)
```

```
['Tennis']
```

```
In [8]: prob_y_pred = dtc.predict_proba(X_test)
print(prob_y_pred)
```

```
[[0.25 0. 0.75]]
```

Question 6 [6 pts BONUS] Informed Search

Consider the search graph above where G is the goal state, and A is the start. Numbers on the edges

represent the cost of the move. Also consider the suggested heuristic $h_1(n)$.

a) [2pts] Is $h_1(n)$ consistent?

b) [2 pts] Run Algorithm A with this heuristic. Show the intermediate steps and the found path.

Node	A	B	C	D	E	F	G
h1 (.)	10	12	10	8	1	4.5	0

Now consider the following new heuristic h2(n):

Node	A	B	C	D	E	F	G
h2 (.)	10	?	9	7	1.5	4.5	0

c) [1pts] Which range of values of h2(B) results in h2() being admissible?

d) [1 pts] which range of values of h2(B) results in h2() being consistent?

a)

To check if a heuristic is consistent, we can apply the triangular inequality:

$$h(n) \leq c(n, n') + h(n')$$

If this condition holds for all edges, then the heuristic is consistent.

Let's consider the edge (B, C):

$$h(B) = 12$$

$$c(B, C) = 1$$

$$h(C) = 10$$

In this case, we find that h(B) is not $\leq c(B, C) + h(C)$ (12 is not ≤ 11).

Therefore, the heuristic is inconsistent.

b)

In the iterations:

Iteration 0: (A, 10)

Iteration 1: (A->B, 1+12=13), (A->C, 4+10=14)

Iteration 2: (A->B->C, 1+1+10=12), (A->B->D, 1+5+8=14), (A->C, 4+10=14)

Iteration 3: (A->B->C->D, 1+1+3+8=13), (A->B->D, 1+5+8=14), (A->C, 4+10=14)

Iteration 4: (A->B->C->D->F, 1+1+3+3+4.5=12.5), (A->B->C->D->E, 1+1+3+8+1=14),

(A->B->C->D->G, 1+1+3+9=14), (A->B->D, 1+5+8=14), (A->C, 4+10=14)

Iteration 5: (A->B->C->D->F->G, 1+1+3+3+5=13), (A->B->C->D->E, 1+1+3+8+1=14),

(A->B->C->D->G, 1+1+3+9=14), (A->B->D, 1+5+8=14), (A->C, 4+10=14)

Thus, the final path is A->B->C->D->F->G.

c)

In order for $h_2(B)$ to be admissible, it must satisfy the condition:

$$h_2(B) \leq h^*(B)$$

where $h^*(B)$ represents the actual cost from B to G.

Given that $h^*(B)$ is equal to 12, the condition becomes:

$$h_2(B) \leq 12$$

Therefore, to ensure admissibility, $h_2(B)$ must be less than or equal to 12.

d)

For $h_2(B)$ to be consistent, it must satisfy the condition:

$$h_2(B) \leq c(B, n) + h(n)$$

where (B, n) represents an edge from B to another node n.

Considering the following edges:

$$(B, A): h_2(B) \leq c(B, A) + h(A) = 1 + 10 = 11$$

$$(B, C): h_2(B) \leq c(B, C) + h(C) = 11$$

$$(B, D): h_2(B) \leq c(B, D) + h(D) = 13$$

From the above conditions, the minimum value of $h_2(B)$ is 11. Therefore, for $h_2(B)$ to be consistent, it should be less than or equal to 11.

Question 7) [6 pts BONUS] Our version of Tic-Tac-Tao

Assume this version of Tic-Tac-Tao. A player can decide (has the option) to play one or two rounds consecutively. However, if he/she plays two rounds, the opponent can also select to play one or two rounds. So, if Max plays two consecutive rounds, Min can play one or two rounds if she wants.

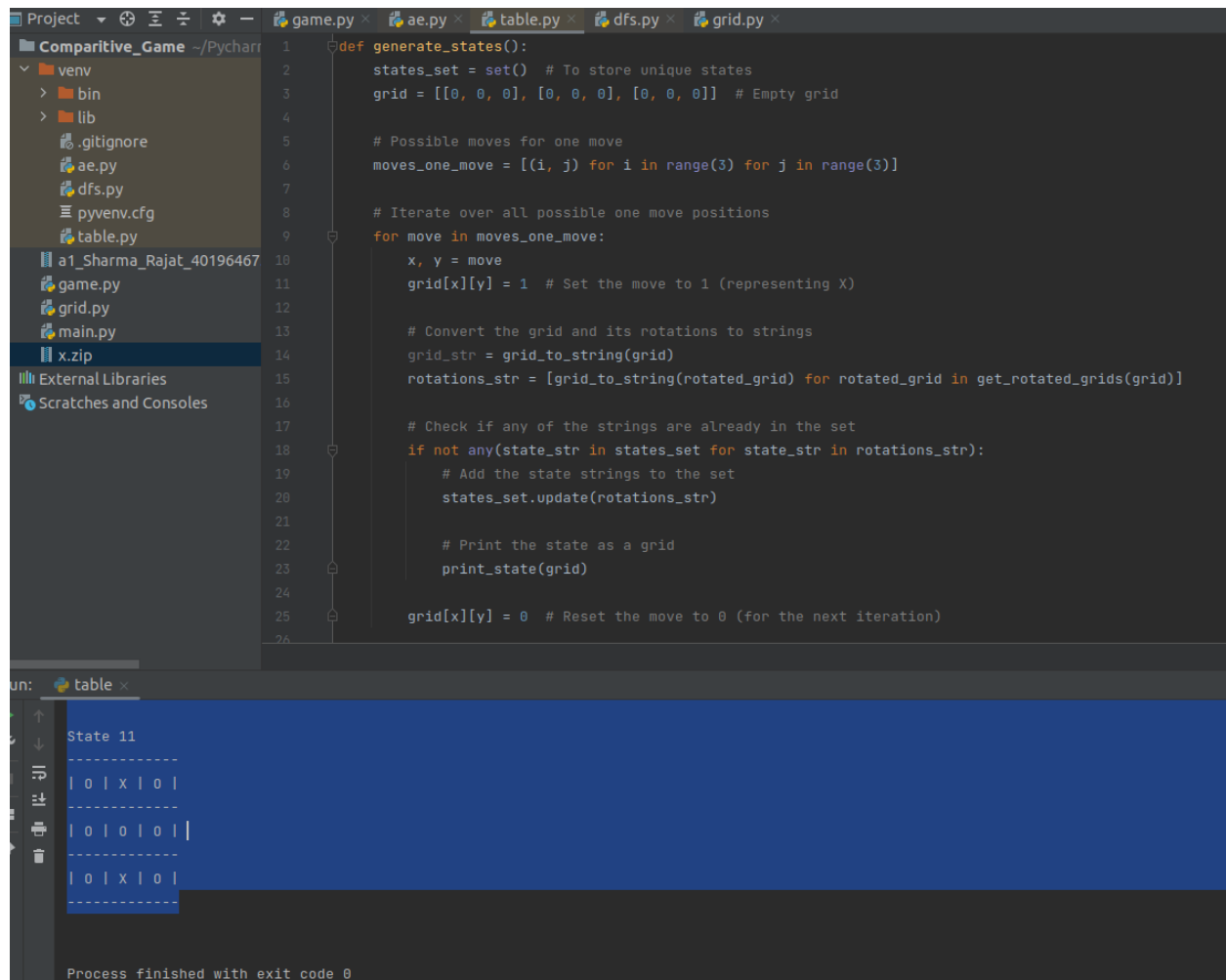
(Board is symmetric, so only the non-symmetric states for sections b and c are enough. For example, with a single move, we had only these 3 non-symmetric states).

[illegible]

In this version of Tic-Tac-Toe, the state can be represented by a 3x3 matrix, where each element can take the value of 0 or 1. A value of 1 represents an 'X' (Max's move), and a value of 0 represents an 'O' (Min's move) in that position.

b)

Wrote a python code to print all possible states



```
1 def generate_states():
2     states_set = set() # To store unique states
3     grid = [[0, 0, 0], [0, 0, 0], [0, 0, 0]] # Empty grid
4
5     # Possible moves for one move
6     moves_one_move = [(i, j) for i in range(3) for j in range(3)]
7
8     # Iterate over all possible one move positions
9     for move in moves_one_move:
10        x, y = move
11        grid[x][y] = 1 # Set the move to 1 (representing X)
12
13        # Convert the grid and its rotations to strings
14        grid_str = grid_to_string(grid)
15        rotations_str = [grid_to_string(rotated_grid) for rotated_grid in get_rotated_grid(grid)]
16
17        # Check if any of the strings are already in the set
18        if not any(state_str in states_set for state_str in rotations_str):
19            # Add the state strings to the set
20            states_set.update(rotations_str)
21
22            # Print the state as a grid
23            print_state(grid)
24
25        grid[x][y] = 0 # Reset the move to 0 (for the next iteration)
```

State 11

```
-----
| 0 | X | 0 |
| 0 | 0 | 0 |
| 0 | X | 0 |
-----
```

Process finished with exit code 0

State 1

```
-----
| X | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
-----
```

State 2

```
-----
| 0 | X | 0 |
```

```
-----  
| 0 | 0 | 0 |  
-----  
| 0 | 0 | 0 |  
-----
```

State 3

```
-----  
| 0 | 0 | 0 |  
-----  
| 0 | x | 0 |  
-----  
| 0 | 0 | 0 |  
-----
```

State 4

```
-----  
| x | x | 0 |  
-----  
| 0 | 0 | 0 |  
-----  
| 0 | 0 | 0 |  
-----
```

State 5

```
-----  
| x | 0 | x |  
-----  
| 0 | 0 | 0 |  
-----  
| 0 | 0 | 0 |  
-----
```

State 6

```
-----  
| x | 0 | 0 |  
-----  
| 0 | x | 0 |  
-----  
| 0 | 0 | 0 |
```

State 7

| x | 0 | 0 |

| 0 | 0 | x |

| 0 | 0 | 0 |

State 8

| x | 0 | 0 |

| 0 | 0 | 0 |

| 0 | 0 | x |

State 9

| 0 | x | 0 |

| x | 0 | 0 |

| 0 | 0 | 0 |

State 10

| 0 | x | 0 |

| 0 | x | 0 |

| 0 | 0 | 0 |

State 11

	0		X		0	

	0		0		0	

	0		X		0	

c) The branching factor for the first round can be calculated as follows:

- Decision to draw one or two symbols: There are 2 options (1 or 2).
- Selecting the cells: There are 9 empty cells initially, so for each option, there are 9 possible cell selections.

Total branching factor = 2 (decision options) * 9 (cell selections) = 18

Therefore, the branching factor for the first round of this version of Tic-Tac-Toe is 18.