Question 1

(a) Yes, the heuristic is admissable. A heuristic is admissable if the heuristic value of a state is less than or equal to minimum cost to reach the goal from the current state. Since there is no node n where the minimum cost to reach the goal is more than the heuristic value, the heuristic is admissable.

Node n	Minimum cost to reach goal from n	H(n)
S	14	13
A	9	6
B	10	9
C	10	8
D	7	5
E	6	4
F	7	4
H	3	1

(b) Algorithm: DFS

Step	Open List (Stack)	POP	Nodes to add
1	S	S	A^S, B^S, C^S
2	A^S, B^S, C^S	A^S	D^A, E^A
3	D^A, E^A, B^S, C^S	D^A	H^D
4	H^D, E^A, B^S, C^S	H^D	G^H
5	G^H, E^A, B^S, C^S	G^H	

Path: $S \rightarrow A \rightarrow D \rightarrow H \rightarrow G$, Cost: 16

Algorithm: A^* Search

NOTE: Subscript represents f-value

Step	Open List (Priority Queue)	DEQUEUE	Nodes to add
1	S	S	$A_{12}^S, B_{13}^S, C_{17}^S$
2	$A_{12}^S, B_{13}^S, C_{17}^S$	A_{12}^S	D_{13}^A, E_{14}^A
3	$B_{13}^S, D_{13}^A, E_{14}^A, C_{17}^S$	B_{13}^S	A_{11}^B, C_{13}^B
4	$A_{11}^B, C_{13}^B, D_{13}^A, E_{14}^A$	A_{11}^B	D_{12}^A, E_{13}^A
5	$D_{12}^A, C_{13}^B, E_{13}^A$	D_{12}^A	E_{12}^D, H_{13}^D
6	$E_{12}^D, C_{13}^B, H_{13}^D$	E_{12}^D	H_{12}^E
7	H_{12}^E, C_{13}^B	H_{12}^E	G_{14}^H
8	C_{13}^B, G_{14}^H	C_{13}^B	F_{13}^C
9	F_{13}^C, G_{14}^H	F_{13}^C	
10	G_{14}^H	G_{14}^H	

Path: $S \rightarrow B \rightarrow A \rightarrow D \rightarrow E \rightarrow H \rightarrow G$, Cost: 14

Question 2

(A) State:

(x,y) tuple where x is the row index and y is the column index of the position of the agent in the maze. The maze can be stored using a 7x7 2D array in the computer, where the first index represents x and second index represents y. Each cell stores the property of the position in the maze (whether it is a start, goal, white or black cell). A priority queue can be used for the storage of states as it allows states to be retrieved based on the lowest f-value for A* search in (B).

Justification:

- Each cell in the maze corresponds to a unique (x, y) tuple which can be easily mapped to the indices in a 2D array.
- When a 2D array is used the current state of each cell is accessible in constant time.
- The operations North, South, East, West can be done by just incrementing and decrementing the respective x or y indices.
- The 2D array also respects the boundary conditions. The agent will not be allowed to move outside the the array's bounds naturally which respects the rules of the maze.

Actions:

- Before all moves, check if successor state coordinates are within maze boundaries and property stored is not black.
- Move North (up) by decreasing the current cell's y value by 1.
- Move South (down) by increasing the current cell's y value by 1.
- Move East (left) by increasing the current cell's x value by 1.
- Move West (right) by decreasing the current cell's x value by 1.

Cost: 1 for every move.

Successor State:

From the current state (x,y),

- For move to North, successor state will be (x, y -1)
- For move to South, successor state will be (x, y+1)
- For move to East, successor state will be (x+1, y)
- For move to West, successor state will be (x-1, y)

Objective:

To reach the goal state G from the start state S using the fewest actions possible.

(B) Since the actions are only North, South, East or West for movement, diagonal movement is not possible so the Manhattan distance is a suitable admissable heuristic. The sum of absolute differences in the x and y coordinates of the current state and the goal state will never overestimate the true distance. It is also consistent as as the current state moves closer to the goal state, the heuristic estimate decreases monotonically. Hence, it provides enough information and at the same time is easy to compute, making it a suitable admissable heuristic.

(C) NOTE: Subscript represents f-value.
Assume that East move takes priority over South move in the event of a tie.

Step	Open List (Priority Queue)	DEQUEUE	Nodes to add
1	S	S	$(0,1)_{14}^{S}$
2	$(0,1)_{14}^{S}$	$(0,1)_{14}^{S}$	$(1,1)_{14}^{(0,1)}$
3	$(1,1)_{14}^{(0,1)}$	$(1,1)_{14}^{(0,1)}$	$(2,1)_{14}^{(1,1)}, (1,2)_{14}^{(1,1)}$
4	$(2,1)_{14}^{(1,1)}, (1,2)_{14}^{(1,1)}$	$(2,1)_{14}^{(1,1)}$	$(2,2)_{14}^{(2,1)}$
5	$(2,2)_{14}^{(2,1)}, (1,2)_{14}^{(1,1)}$	$(2,2)_{14}^{(2,1)}$	$(3,2)_{14}^{(2,2)}$

Question 3

(a)

3	-0.20	0.217	0.628	Food
2	-0.48	blocked	-0.02	Tiger
1	-0.7166	-0.71	-0.4304	-0.75
	1	2	3	4

$$Q^{t+1}(s,a) = \sum_{s'} Pr(s'|s,a)[R(s,a,s') + \gamma V^{t}(s')]$$

$$V^{t+1}(s) = \max_{a} Q^{t+1}(s,a)$$

$$\begin{aligned} Q^{t+1}((1,1),N) &= 0.6 \times [-0.2 + 0.9 \times -0.48] + 0.2 \times [-0.2 + 0.9 \times -0.71] \\ &\quad + 0.2 \times [-0.2 + 0.9 \times -0.72] = -0.7166 \\ Q^{t+1}((1,1),E) &= 0.6 \times [-0.2 + 0.9 \times -0.71] + 0.2 \times [-0.2 + 0.9 \times -0.48] \\ &\quad + 0.2 \times [-0.2 + 0.9 \times -0.72] = -0.7994 \\ Q^{t+1}((1,1),S) &= 0.6 \times [-0.2 + 0.9 \times -0.72] + 0.2 \times [-0.2 + 0.9 \times -0.71] \\ &\quad + 0.2 \times [-0.2 + 0.9 \times -0.72] = -0.8462 \\ Q^{t+1}((1,1),W) &= 0.6 \times [-0.2 + 0.9 \times -0.72] + 0.2 \times [-0.2 + 0.9 \times -0.48] \\ &\quad + 0.2 \times [-0.2 + 0.9 \times -0.72] = -0.8048 \end{aligned}$$

$$V^{t+1}((1,1)) = -0.7166$$

$$\begin{aligned} Q^{t+1}((1,3),N) &= 0.6 \times [-0.2 + 0.9 \times -0.02] + 0.2 \times [-0.2 + 0.9 \times -0.75] \\ &\quad + 0.2 \times [-0.2 + 0.9 \times -0.47] = -0.4304 \\ Q^{t+1}((1,3),E) &= 0.6 \times [-0.2 + 0.9 \times -0.75] + 0.2 \times [-0.2 + 0.9 \times -0.02] \\ &\quad + 0.2 \times [-0.2 + 0.9 \times -0.47] = -0.6932 \\ Q^{t+1}((1,3),S) &= 0.6 \times [-0.2 + 0.9 \times -0.47] + 0.2 \times [-0.2 + 0.9 \times -0.75] \\ &\quad + 0.2 \times [-0.2 + 0.9 \times -0.71] = -0.7166 \\ Q^{t+1}((1,3),W) &= 0.6 \times [-0.2 + 0.9 \times -0.71] + 0.2 \times [-0.2 + 0.9 \times -0.02] \\ &\quad + 0.2 \times [-0.2 + 0.9 \times -0.47] = -0.6716 \end{aligned}$$

$$V^{t+1}((1,3)) = -0.4304$$

(b)

3	E	0.217	E	Food
2	N	blocked	N	Tiger
1	N	E	N	W
	1	2	3	4

From part (a),

$$\pi^*((1,1)) = \arg\max_a Q^{t+1}((1,1),a) = N$$
$$\pi^*((1,3)) = \arg\max_a Q^{t+1}((1,3),a) = E$$

Question 4

(a)

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
nltk.download('punkt')

import numpy as np
import pandas as pd
from gensim.models import word2vec

from google.colab import drive
drive.mount('/content/drive')

import re # For regular expressions
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
def load_data():
    """ Read tweets from the file.
        Return:
            list of lists (list_words), with words from each of the processed tweets
    """
    tweets = pd.read_csv('/content/drive/MyDrive/Corona_Tweets_large.csv', header=None)
    list_words = []
    # define stop words
    stop_words = set(stopwords.words('english'))
    ### iterate over all tweets from the dataset
    for i in tweets.index:
      tweet = tweets.loc[i, 0]
      # Remove URLs
      text = re.sub(r'https?://\S+', '', tweet)
      ### remove non-letter.
      text = re.sub(r'[^a-zA-Z]', ' ', text)
      ### tokenize
      words = nltk.word_tokenize(text)

      new_words = []
      ### iterate over all words of a tweet, remove the stop words and convert a word (w) to the lower case
      for w in words:
        w_lower = w.lower()
        if w_lower not in stop_words:
          new_words.append(w_lower)

      list_words.append(new_words)
    return list_words

# check a few samples of twitter corpus
twitter_corpus = load_data()
print(twitter_corpus[:3])
```

```
[['smelled', 'scent', 'hand', 'sanitizers', 'today', 'someone', 'past', 'would', 'think', 'intoxicated'], ['hey', 'yankees', 'yankeespr', 'mlb', 'made',
```

(b)

```
# Creating the word2vec model and setting values for the various parameters

# Function for model training

def train_word2vec(twitter_corpus, num_features, context):
  num_workers = 4     # Number of parallel threads, can be changed
  min_word_count = 10  # Minimum word count. You can change it also.
  downsampling = 1e-3 # (0.001) Downsample setting for frequent words, can be changed

  # Initializing the train model
  print(f"Training Word2Vec model with {num_features} features and {context} context window size....")
  model = word2vec.Word2Vec(twitter_corpus, workers=num_workers,\
                            vector_size=num_features,\
                            min_count=min_word_count,\
                            window=context,
                            sample=downsampling)

  # To make the model memory efficient
  model.init_sims(replace=True)

  return model

settings = [(20, 5), (50, 10), (75, 15)]
models = []

for num_features, context in settings:
    model = train_word2vec(twitter_corpus, num_features, context)
    models.append(model)

print("All models trained")
```

```
Training Word2Vec model with 20 features and 5 context window size....
<ipython-input-110-2896e80c6a99>:19: DeprecationWarning: Call to deprecated `init_sims` (Gensim 4.0.0 implemented internal optimizations that make calls to
  model.init_sims(replace=True)
WARNING:gensim.models.keyedvectors:destructive init_sims(replace=True) deprecated & no longer required for space-efficiency
Training Word2Vec model with 50 features and 10 context window size....
WARNING:gensim.models.keyedvectors:destructive init_sims(replace=True) deprecated & no longer required for space-efficiency
Training Word2Vec model with 75 features and 15 context window size....
WARNING:gensim.models.keyedvectors:destructive init_sims(replace=True) deprecated & no longer required for space-efficiency
All models trained
```

```
for model, (num_features, context) in zip(models, settings):
    print(f"Model with {num_features} features and a context window of {context}:")
    print(model.wv.most_similar("covid"))
```

```
Model with 20 features and a context window of 5:
[('coronavirus', 0.880866527557373), ('current', 0.7242090702056885), ('repor', 0.6912548542022705), ('abroad', 0.6783382892608643), ('unknown', 0.672848939
Model with 50 features and a context window of 10:
[('coronavirus', 0.4255777299404144 3), ('concerning', 0.4255542755126953), ('infe', 0.4168890118598938), ('races', 0.413379430770874), ('mln', 0.41326448321
Model with 75 features and a context window of 15:
[('infe', 0.466819167137146), ('infecti', 0.4321218729019165), ('climbed', 0.4164325892925262 5), ('incr', 0.407633900642395), ('mln', 0.4061895608901977 5),
```

+ Code + Text

```
for model, (num_features, context) in zip(models, settings):
    print(f"Model with {num_features} features and a context window of {context}:")
    print(model.wv.most_similar("grocery"))
```

```
Model with 20 features and a context window of 5:
[('shopping', 0.9466589689254761), ('store', 0.9262773394584656), ('shop', 0.9156877398490906), ('buses', 0.8945198059082031), ('dining', 0.8872543573379517
Model with 50 features and a context window of 10:
[('shopping', 0.9162435531616211), ('shop', 0.9002662301063538), ('store', 0.898946225643158), ('stores', 0.8803632855415344), ('restaurant', 0.864786207675
Model with 75 features and a context window of 15:
[('shop', 0.9381406903266907), ('store', 0.9290154576301575), ('stores', 0.9211677312850952), ('shopping', 0.903228223323822), ('pack', 0.8588300347328186),
```

```
for model, (num_features, context) in zip(models, settings):
    print(f"Model with {num_features} features and a context window of {context}:")
    print(model.wv.most_similar("virus"))
```

```
Model with 20 features and a context window of 5:
[('deadly', 0.7818624973297119), ('contagious', 0.7509030699729919), ('control', 0.7446085810661316), ('viruses', 0.7289645671844482), ('uncontrolled', 0.72
Model with 50 features and a context window of 10:
[('deadly', 0.6614598035812378), ('contagious', 0.6559390425682068), ('controlled', 0.6473415493965149), ('wuhan', 0.6172404289245605), ('mutation', 0.60287
Model with 75 features and a context window of 15:
[('deadly', 0.6631927490234375), ('controlled', 0.6271705627441406), ('herd', 0.6147304177284241), ('mutation', 0.6085119247436523), ('contagious', 0.607611
```

```
for model, (num_features, context) in zip(models, settings):
    print(f"Model with {num_features} features and a context window of {context}:")
    print(model.wv.most_similar("corona"))
```

```
Model with 20 features and a context window of 5:
[('coronainfoch', 0.7689714431762695), ('worldnews', 0.7321677803993225), ('source', 0.7146664261817932), ('utc', 0.7091984748840332), ('shalinitelevision',
Model with 50 features and a context window of 10:
[('coronainfoch', 0.702916145324707), ('worldnews', 0.6888173222541809), ('source', 0.60324627161026), ('coronainindia', 0.5926268100738525), ('utc', 0.5862
Model with 75 features and a context window of 15:
[('coronainfoch', 0.692014217376709), ('worldnews', 0.6426941156387329), ('source', 0.5553585290908813), ('bat', 0.5439143180847168), ('korona', 0.536787092
```

```
for model, (num_features, context) in zip(models, settings):
    print(f"Model with {num_features} features and a context window of {context}:")
    print(model.wv.most_similar("pandemic"))
```

```
Model with 20 features and a context window of 5:
[('crisis', 0.8270560503005981), ('disruption', 0.7848400473594666), ('midst', 0.783033549785614), ('pan', 0.7794546484947205), ('wemerry', 0.77500939369201
Model with 50 features and a context window of 10:
[('crisis', 0.7169919610023499), ('pandemi', 0.6076443791389465), ('pande', 0.6002398729324341), ('pan', 0.5887707471847534), ('disruption', 0.5620629787445
Model with 75 features and a context window of 15:
[('crisis', 0.7158800959587097), ('downturns', 0.6055337190628052), ('disruption', 0.5616447329521179), ('pan', 0.551657140254974 4), ('crises', 0.5516414642
```

```
# function to find odd word out
def odd_one_out(model, words):
  odd_word = model.wv.doesnt_match(words)
  return odd_word;

# list of test words
words = ["covid", "grocery", "virus", "corona", "pandemic"]

for model, (num_features, context) in zip(models, settings):
    print(f"Model with {num_features} features and a context window of {context}:")
    print(f"The odd word out is: {odd_one_out(model, words)}")
```

```
Model with 20 features and a context window of 5:
The odd word out is: grocery
Model with 50 features and a context window of 10:
The odd word out is: grocery
Model with 75 features and a context window of 15:
The odd word out is: grocery
```

Observations:

I trained 3 models, with (num_features, context) values (20, 5), (50, 10) and (75, 15). First I tried finding similar words using each model. From the tests, I can concude that (50, 10) is the best set of hyperparameters for this dataset. When a similar word to “covid” was searched for, (75, 15) returned “infe” as the most similar word while the other 2 models returned “coronavirus” as the most similar word, which was not in the top 5 similar words for (75, 15). Models (20, 5) and (50, 10) are give similar output for the top similarity words, but for the word “lockdown” (20, 5) has “tourist” as its second most similar word, which is intuitively less similar to “lockdown” than words like “easing” or “imposed” or “lifted”, giving (50, 10) the edge.

Secondly, I tried finding the odd word out using the 3 models and the 5 test words provided in All models performed well and output the intuitive odd word out “grocery”.

Question 5

(a) State space size: 16 (4x4 map),
Action space size: 4 (agent can move UP, DOWN, LEFT, RIGHT)

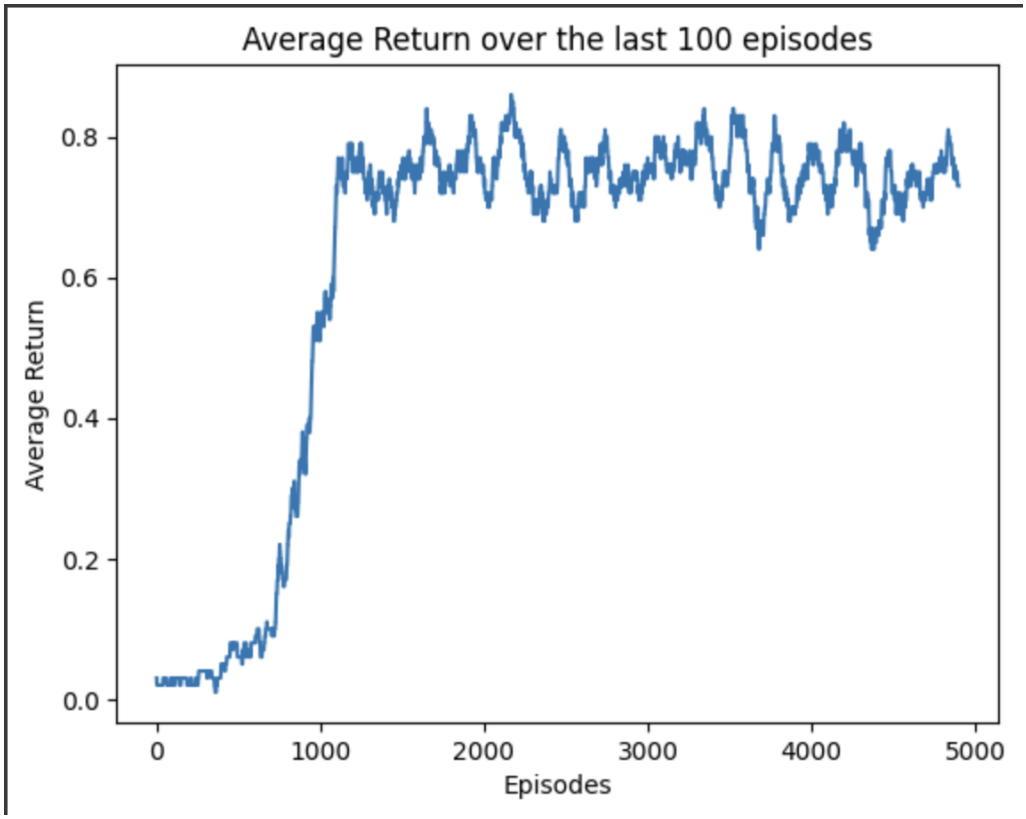
```python
import numpy as np
import gym
import matplotlib.pyplot as plt

env = gym.make('FrozenLake-v1', desc=None, map_name="4x4", is_slippery=True)

# define q-learning parameters
learning_rate = 0.5
discount_factor = 0.99
epsilon = 1.0
num_episodes = 5000
epsilon_decay = 0.001

# initialise q-table
q_table = np.zeros([env.observation_space.n, env.action_space.n])

# function to choose next action using epsilon-greedy policy
def choose_action(state, Q, epsilon):
    if np.random.random() < epsilon:
        action = env.action_space.sample()  # exploration
    else:
        action = np.argmax(q_table[state]) # exploitation
    return action

# function to update the q value based on the action taken and perceived reward
def update_q_table(state, new_state, reward, action, learning_rate, discount_factor):
    q_table[state, action] = q_table[state, action] + learning_rate * (reward + discount_factor * np.max(q_table[new_state]) - q_table[state, action])
```

(b)

```python
# initialise list to store total rewards per episode
rewards_per_episode = []

for episode in range(num_episodes):
  state = env.reset() # reset environment for new episode
  total_rewards = 0
  done = False

  while not done:
    action = choose_action(state, q_table, epsilon) # choose action based on policy
    new_state, reward, done, info = env.step(action) # take action
    total_rewards += reward # add reward to total_rewards
    update_q_table(state, new_state, reward, action, learning_rate, discount_factor) # update q table
    state = new_state

  rewards_per_episode.append(total_rewards)
  epsilon = max(epsilon - epsilon_decay, 0)

print(q_table)
```

```
[[0.50095108 0.28479098 0.28287162 0.28186629]
 [0.19022654 0.15296049 0.15640963 0.56249411]
 [0.14118628 0.20350697 0.18242033 0.49805124]
 [0.08257227 0.20531736 0.19317209 0.49679995]
 [0.559944   0.10643156 0.20024762 0.2612365 ]
 [0.         0.         0.         0.        ]
 [0.12110605 0.00874957 0.00725997 0.00902878]
 [0.         0.         0.         0.        ]
 [0.22461895 0.25585841 0.24673634 0.63953766]
 [0.13967717 0.77780908 0.10522435 0.12695241]
 [0.77772217 0.07460199 0.06302066 0.0773512 ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.19326123 0.33092771 0.91156559 0.23846982]
 [0.48774741 0.99498838 0.45892984 0.34894832]
 [0.         0.         0.         0.        ]]
```

```python
# calculate average returns over each 100 episode
average_return = []
for i in range(len(rewards_per_episode) - 100):
  average_return_100_episodes = np.mean(rewards_per_episode[i:i+100])
  average_return.append(average_return_100_episodes)

print(average_return)

# plot average return over last 100 episodes against episode number
plt.plot(average_return)
plt.xlabel('Episodes')
plt.ylabel('Average Return')
plt.title('Average Return over the last 100 episodes')
plt.show()
```



(c)

I initially set epsilon to 1.0, reducing it by 0.001 in every episode. This is to allow for greater exploration at the start when the agent has no prior knowedge. As the agent gathers information, the epsilon decay causes the epsilon to drop, reducing the amount of exploration as the agent gains more knowledge about the environment. If a high epsilon of 0.5 is used constantly, it means that the agent is still taking random actions half the time even in later episodes so it might take longer for the Q-table to reach convergence to optimal values and could also lead to a suboptimal policy.