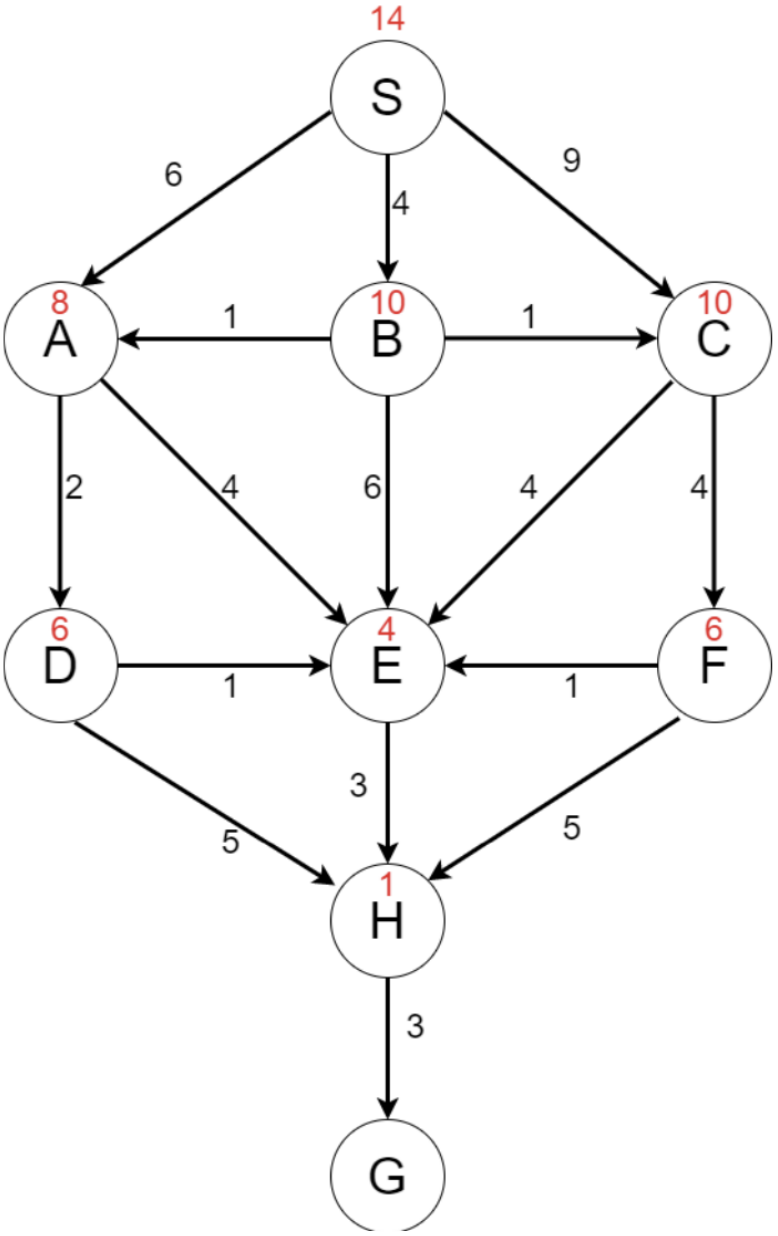
Question 1 [10 points]



For the above graph (heuristic values are provided in red color and actual costs are in black color), please provide answers to the following answers:

Please note that S is start state and G is goal state.

(a) Is the heuristic admissible? Provide justification.

Node n	Minimum cost to reach goal from n	h(n)
S	14 (S -> B -> A -> D -> E ->H	14

	-> G)	
A	9 (A -> D -> E -> H -> G)	8
B	10 (B -> A -> D -> E -> H -> G)	10
C	10 (C -> E -> H -> G)	10
D	7 (D -> E -> H -> G)	6
E	6 (E -> H -> G)	4
F	7 (F -> E -> H -> G)	6
H	3 (H -> G)	1

A heuristic is admissible if h(n) ≤ minimum cost(n) for all n. Since this applies for all the nodes, the heuristic is admissible.

(b) Is the heuristic consistent? Provide justification.

Node n	Successor p	c(n, p)	h(p)	h(n)	c(n, p) + h(p)
S	A	6	8	14	14
S	B	4	10	14	14
S	C	9	10	14	19
A	D	2	6	8	8
A	E	4	4	8	8
B	A	1	8	10	9
B	C	1	10	10	11
B	E	6	4	10	10
C	E	4	4	10	8
C	F	4	6	10	10
D	E	1	6	6	7

D	H	5	1	6	6
E	H	3	1	4	4
F	E	1	4	6	5
F	H	5	1	6	6
H	G	3	1	1	4

A heuristic is consistent if

- for every node n, and all its successors p, we have h(n) ≤ c(n, p) + h(p), where the estimated cost of reaching the goal from n is no greater than the step cost of getting to p plus the estimated cost of reaching the goal from p
- h(g) = 0

Since this does not apply for B, C and F, the heuristic is not consistent.

(c) Provide the search steps (as discussed in class) with vanilla **Breadth First Search (BFS)**

Show the **final solution path** and the **cost of that solution** for each algorithm. Also compute the search steps for **A* search** with the following heuristic:

Specify for each algorithm if the open list is queue, stack, or priority queue. As a simplifying assumption, let index zero (i.e. the first element) in the open list be the top of the stack or front of the (priority) queue, as appropriate for the corresponding algorithm. Break ties by alphabetical order.

Algorithm: BFS

Step #	Queue	POP or DEQUEUE	Nodes to add
1	S		
2	A^S, B^S, C^S	S	A^S, B^S, C^S
3	B^S, C^S, D^A, E^A	A^S	D^A, E^A
4	C^S, D^A, E^A	B^S	-
5	D^A, E^A, F^C	C^S	F^C
6	E^A, F^C, H^D	D^A	H^D
7	F^C, H^D	E^A	-
8	H^D	F^C	-

9	G^H	H^D	G^H
10		G^H	

Path: SBADEHG, Cost: 14

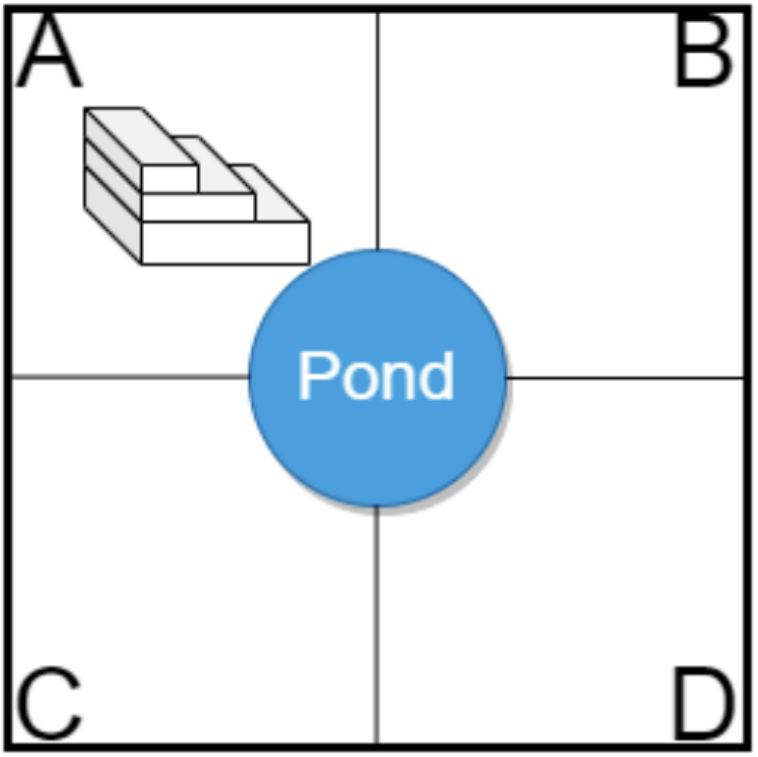
Algorithm: A* Search

Step #	Priority Queue	POP or DEQUEUE	Nodes to add
1	$(S,\ 0 + 14 = 14)$		
2	$(A^S,\ 6 + 8 = 14),$ $(B^S,\ 4 + 10 = 14),$ $(C^S,\ 9 + 10 = 19)$	$(S,\ 0 + 14 = 14)$	$(A^S,\ 6 + 8 = 14),$ $(B^S,\ 4 + 10 = 14),$ $(C^S,\ 9 + 10 = 19)$
3	$(B^S,\ 4 + 10 = 14),$ $(D^A, 8 + 6 = 14),$ $(E^A,\ 10 + 4 = 14),$ $(C^S,\ 9 + 10 = 19)$	$(A^S,\ 6 + 8 = 14)$	$(D^A, 8 + 6 = 14),$ $(E^A,\ 10 + 4 = 14)$
4	$(A^B,\ 5 + 8 = 13),$ $(D^A, 8 + 6 = 14),$ $(E^A,\ 10 + 4 = 14),$ $(E^B,\ 10 + 4 = 14),$ $(C^B,\ 5 + 10 = 15),$ $(C^S,\ 9 + 10 = 19)$	$(B^S,\ 4 + 10 = 14)$	$(A^B,\ 5 + 8 = 13),$ $(C^B,\ 5 + 10 = 15),$ $(E^B,\ 10 + 4 = 14)$
5	$(D^A,\ 7 + 6 = 13),$ $(E^A,\ 9 + 4 = 13),$ $(D^A, 8 + 6 = 14),$ $(E^A,\ 10 + 4 = 14),$ $(E^B,\ 10 + 4 = 14),$ $(C^B,\ 5 + 10 = 15),$ $(C^S,\ 9 + 10 = 19)$	$(A^B,\ 5 + 8 = 13)$	$(D^A,\ 7 + 6 = 13),$ $(E^A,\ 9 + 4 = 13)$
6	$(E^D,\ 8 + 4 = 12),$ $(E^A,\ 9 + 4 = 13),$ $(H^D,\ 12 + 1 = 13),$ $(D^A, 8 + 6 = 14),$ $(E^A,\ 10 + 4 = 14),$ $(E^B,\ 10 + 4 = 14),$ $(C^B,\ 5 + 10 = 15),$	$(D^A,\ 7 + 6 = 13)$	$(E^D,\ 8 + 4 = 12),$ $(H^D,\ 12 + 1 = 13)$

	$(C^S, 9 + 10 = 19)$		
7	$(H^E, 11 + 1 = 12)$, $(E^A, 9 + 4 = 13)$, $(H^D, 12 + 1 = 13)$, $(D^A, 8 + 6 = 14)$, $(E^A, 10 + 4 = 14)$, $(E^B, 10 + 4 = 14)$, $(C^B, 5 + 10 = 15)$, $(C^S, 9 + 10 = 19)$	$(E^D, 8 + 4 = 12)$	$(H^E, 11 + 1 = 12)$
8		$(H^E, 11 + 1 = 12)$	$(G^H, 14 + 0 = 14)$

Path: SBADEHG, Cost: 14

Question 2 [10 points]: Moving boxes around the pond



You are hired by a moving company to move N boxes from a corner of a room to the opposite corner of the room. The room can be roughly divided into four corners, and there is an indoor pond in the middle of the room preventing any direct diagonal crossing. The three boxes have different sizes and can only be stacked in such a way that larger boxes can never be placed on top of smaller ones.

Please answer the following questions:

A. Given N boxes (stacked appropriately in corner A with the smallest box on the top), formulate this problem as a search problem, i.e.,
   - indicate the states (describe specifically what a state is in this problem, how you would store it in a computer using a data structure, and justify the correctness of your state representation).
   - actions,
   - cost of different actions,
   - successor state (for each action) and (**give one example of a state with ALL of its successor states listed**).
   - the objective.

   Please pay special attention to what **data structure** you would use to store any given state in this problem. We are looking for a state representation that is easily stored in a computer, and it is easy to generate its successor for any valid action. For example, in the 8-puzzle problem, a 3x3 numpy array can represent

any state. ***Just an intuitive definition for what is a state will not suffice for this question.***

- States
  - A state in this problem is the configuration of the location of all the boxes in the room.
  - This can be represented as a nested array of length 4, with each corner having an array of length N to represent each box. If the box is located in that corner, it will be represented as 1 in the array, else it will be represented as 0.
  - We can easily check and update the location of each box in O(1) time using the index of the array e.g. arr[0][0] for corner A, box 1.
- Actions
  - Each action would be represented in the same nested array format. Each action will involve the moving of 1 box. The "TO" location of the box will be represented as 1, and the "FROM" location of the box will be represented as -1, and the lack of movement of the rest of the boxes will be represented as 0.
- Cost of different actions
  - As each action will involve the movement of 1 box, each action will have a cost of 1.
- Successor state (for each action)
  - The successor state will be the configuration of the location of all the boxes in the room after an action has been taken.
  - For example, using the starting state:
    - Starting state: all the boxes are at corner A
      - [[1, 1, 1], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
    - Successor state 1: move top box to corner B
      - [[0, 1, 1], [1, 0, 0], [0, 0, 0], [0, 0, 0]]
    - Successor state 2: move top box to corner C
      - [[0, 1, 1], [0, 0, 0], [1, 0, 0], [0, 0, 0]]
- Objective
  - The objective is to move all the boxes to the opposite corner of the room. Hence, the goal state is [[0, 0, 0], [0, 0, 0], [0, 0, 0], [1, 1, 1]]

B. For using heuristic search methods such as A*, provide an admissible heuristic and justify why it is admissible.

Note: Make sure that heuristic is easy to compute, informative (i.e., do not assume heuristic value as zero for all the states) and intuitive enough so that you can justify that it is admissible.

The heuristic would be the number of boxes that are not yet at the goal. A heuristic is admissible if $h(n) \leq$ minimum cost(n) for all n. This heuristic is admissible because the best case would be that each box incurs 1 cost to move to the goal, hence the total cost incurred would be equal to the number of boxes.

C. For the case where $N = 3$ (represented in the picture above), compute the first three steps of **DFS search**. You can use the table structure as in question 1c to show different steps.

Algorithm: DFS search

Step #	Stack	POP or DEQUEUE	Nodes to add
1	[[1, 1, 1], [0, 0, 0], [0, 0, 0], [0, 0, 0]]		
2	Move box 1 to corner B [[0, 1, 1], [1, 0, 0], [0, 0, 0], [0, 0, 0]] Move box 1 to corner C [[0, 1, 1], [0, 0, 0], [1, 0, 0], [0, 0, 0]]	[[1, 1, 1], [0, 0, 0], [0, 0, 0], [0, 0, 0]]	
3	Move box 2 to corner C [[0, 0, 1], [1, 0, 0], [0, 1, 0], [0, 0, 0]] Move box 1 to corner D [[0, 1, 1], [0, 0, 0], [0, 0, 0], [1, 0, 0]] [[0, 1, 1], [0, 0, 0], [1, 0, 0], [0, 0, 0]]	[[0, 1, 1], [1, 0, 0], [0, 0, 0], [0, 0, 0]]	

D. Give a solution to the problem for $N = 3$. You can achieve it either using some methods from the class or through guessing.

Algorithm: Best First Search

Step #	Stack	POP or DEQUEUE	Nodes to add
1	([[1, 1, 1], [0, 0, 0], [0, 0, 0], [0, 0, 0]], 3)		
2	([[0, 1, 1], [1, 0, 0], [0, 0, 0], [0, 0, 0]], 3) ([[0, 1, 1], [0, 0, 0], [1, 0, 0], [0, 0, 0]], 3)	([[1, 1, 1], [0, 0, 0], [0, 0, 0], [0, 0, 0]], 3)	([[0, 1, 1], [1, 0, 0], [0, 0, 0], [0, 0, 0]], 3) ([[0, 1, 1], [0, 0, 0], [1, 0, 0], [0, 0, 0]], 3)
3	([[0, 1, 1], [0, 0, 0], [0, 0, 0], [1, 0, 0]], 2)	([[0, 1, 1], [1, 0, 0], [0, 0, 0], [0, 0, 0]], 3)	([[0, 1, 1], [0, 0, 0], [0, 0, 0], [1, 0, 0]], 2)

	([[0, 0, 1], [1, 0, 0], [0, 1, 0], [0, 0, 0]], 3) ([[0, 1, 1], [0, 0, 0], [1, 0, 0], [0, 0, 0]], 3)		([[0, 0, 1], [1, 0, 0], [0, 1, 0], [0, 0, 0]], 3)
4	([[0, 0, 1], [0, 1, 0], [0, 0, 0], [1, 0, 0]], 2) ([[0, 0, 1], [0, 0, 0], [0, 1, 0], [1, 0, 0]], 2) ([[0, 1, 1], [0, 0, 0], [1, 0, 0], [0, 0, 0]], 3) ([[0, 0, 1], [1, 0, 0], [0, 1, 0], [0, 0, 0]], 3)	([[0, 1, 1], [0, 0, 0], [0, 0, 0], [1, 0, 0]], 2)	([[0, 0, 1], [0, 1, 0], [0, 0, 0], [1, 0, 0]], 2) ([[0, 0, 1], [0, 0, 0], [0, 1, 0], [1, 0, 0]], 2)
5	([[0, 0, 1], [0, 0, 0], [0, 0, 0], [1, 1, 0]], 1) ([[0, 0, 0], [0, 1, 1], [0, 0, 0], [1, 0, 0]], 2) ([[0, 0, 0], [0, 1, 0], [0, 0, 1], [1, 0, 0]], 2) ([[0, 0, 1], [0, 0, 0], [0, 1, 0], [1, 0, 0]], 2) ([[0, 1, 1], [0, 0, 0], [1, 0, 0], [0, 0, 0]], 3) ([[0, 0, 1], [1, 0, 0], [0, 1, 0], [0, 0, 0]], 3)	([[0, 0, 1], [0, 1, 0], [0, 0, 0], [1, 0, 0]], 2)	([[0, 0, 1], [0, 0, 0], [0, 0, 0], [1, 1, 0]], 1) ([[0, 0, 0], [0, 1, 1], [0, 0, 0], [1, 0, 0]], 2) ([[0, 0, 0], [0, 1, 0], [0, 0, 1], [1, 0, 0]], 2)
6	([[0, 0, 0], [0, 0, 1], [0, 0, 0], [1, 1, 0]], 1) ([[0, 0, 0], [0, 0, 0], [1, 0, 0], [1, 1, 0]], 1) ([[0, 0, 0], [0, 1, 1], [0, 0, 0], [1, 0, 0]], 2) ([[0, 0, 0], [0, 1, 0], [0, 0, 1], [1, 0, 0]], 2) ([[0, 0, 1], [0, 0, 0], [0, 1, 0], [1, 0, 0]], 2) ([[0, 1, 1], [0, 0, 0], [1, 0, 0], [0, 0, 0]], 3)	([[0, 0, 1], [0, 0, 0], [0, 0, 0], [1, 1, 0]], 1)	([[0, 0, 0], [0, 0, 1], [0, 0, 0], [1, 1, 0]], 1) ([[0, 0, 0], [0, 0, 0], [1, 0, 0], [1, 1, 0]], 1)

	([[0, 0, 1], [1, 0, 0], [0, 1, 0], [0, 0, 0]], 3)		
7		([[0, 0, 0], [0, 0, 1], [0, 0, 0], [1, 1, 0]], 1)	([[0, 0, 0], [0, 0, 0], [0, 0, 0], [1, 1, 1]], 0)

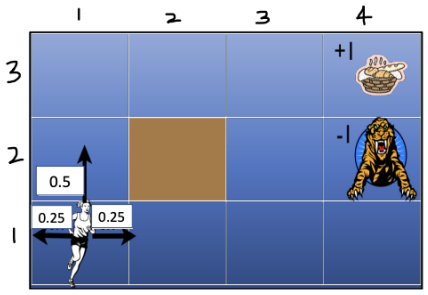
E. How would the state representation in part A change if the following modifications are applied to the problem:

- The cost to move one box is now proportional to the size of the box (cost of 1 for moving the smallest box, cost of 3 for moving the largest one).
- If the moving company employee moves from one corner to the other without carrying any box, it also incurs a cost of 0.5

1. We can have an array which holds the cost of moving each box for all the states, and a variable to keep track of the current cost incurred to move the boxes thus far for each state.
2. We can add an additional variable to keep track of which corner the moving company employee is at, such as (x, y).

Question 3 [10 Points]

Please recall the robot runner in the grid world example with tiger and food cells on right last column.



The states are grid cells, i.e., (1,1), (1,2), …. First component of a state is the row number, second component is the column number. The actions available are North, East, West, South.

Rewards are given as follows:

- R(*, *, *) = -0.1 (Penalty of movement, assuming resulting state is not food or tiger)
- R(*, *, (3,4)) = +2 (Moving to Food cell)
- R(*, *, (2,4)) = -1 (Moving to Tiger cell)

Transition probability is 0.5 to the state in the direction of the action and 0.25 in states perpendicular to the direction of the action. If agent hits a wall, the agent moves back to its original location. Terminating states are when agent is in the food cell or in the tiger cell. Discount factor is 0.95.

For value iteration method, the values of different states $V^t(.)$ at an iteration **t** are given by:

3	0.653	1.059	1.381	**Food**
2	0.400		0.434	**Tiger**
1	0.082	-0.11	-0.00	-0.35
	1	**2**	**3**	**4**

(a) Compute the values $V^{t+1}(.)$ for different states at iteration **t+1.** The V^{t+1} values for some states are given in the table below. You need to compute values for states with "?" entry. Write numerical answers for such cells in the table below.

3	0.653	1.059	1.381	**Food**
2	0.400	blocked	0.434	**Tiger**
1	0.082	?	?	?
	1	**2**	**3**	**4**

Please show **analytical expressions** denoting all the computations (**without** using any python code).

s	a	s'	P(s' | s, a)	R(s, a, s')

(1, 2)	N	(1, 2) - N	0.5	-0.1
		(1, 3) - E	0.25	-0.1
		(1, 1) - W	0.25	-0.1
	E	(1, 3) - E	0.5	-0.1
		(1, 2) - N, S	0.5	-0.1
	W	(1, 1) - W	0.5	-0.1
		(1, 2) - N, S	0.5	-0.1
	S	(1, 2) - S	0.5	-0.1
		(1, 3) - E	0.25	-0.1
		(1, 1) - W	0.25	-0.1
(1, 3)	N	(2, 3) - N	0.5	-0.1
		(1, 4) - E	0.25	-0.1
		(1, 2) - W	0.25	-0.1
	E	(1, 4) - E	0.5	-0.1
		(2, 3) - N	0.25	-0.1
		(1, 3) - S	0.25	-0.1
	W	(1, 2) - W	0.5	-0.1
		(2, 3) - N	0.25	-0.1
		(1, 3) - S	0.25	-0.1
	S	(1, 3) - S	0.5	-0.1
		(1, 4) - E	0.25	-0.1
		(1, 2) - W	0.25	-0.1
(1, 4)	N	(2, 4) - N	0.5	-1
		(1, 4) - E	0.25	-0.1
		(1, 3) - W	0.25	-0.1
	E	(1, 4) - E, S	0.75	-0.1
		(2, 4) - N	0.25	-1
	W	(1, 3) - W	0.5	-0.1
		(2, 4) - N	0.25	-1

		(1, 4) - S	0.25	-0.1
	S	(1, 4) - S, E	0.75	-0.1
		(1, 3) - W	0.25	-0.1

$$Q(s,a) = \sum_{s'} P(s' \mid s,\ a)\,[R(s,\ a,\ s') + \gamma V(s')]$$

$$Q((1,\ 2),\ N)$$
$$= P((1,2) \mid (1,\ 2),\ N)\,[R((1,2),\ N,\ (1,2)) + 0.95V^{t}(1,2)]$$
$$+ P((1,3) \mid (1,\ 2),\ N)\,[R((1,2),\ N,\ (1,3)) + 0.95V^{t}(1,3)]$$
$$+ P((1,1) \mid (1,\ 2),\ N)\,[R((1,2),\ N,\ (1,1)) + 0.95V^{t}(1,1)]$$
$$= 0.5\,[-\ 0.1 + 0.95(-\ 0.11)]$$
$$+ 0.25\,[-\ 0.1 + 0.95(-\ 0.00)]$$
$$+ 0.25\,[-\ 0.1 + 0.95(0.082)]$$
$$= -\ 0.132775$$

$$Q((1,\ 2),\ E)$$
$$= P((1,3) \mid (1,\ 2),\ E)\,[R((1,2),\ E,\ (1,3)) + 0.95V^{t}(1,3)]$$
$$+ P((1,1) \mid (1,\ 2),\ E)\,[R((1,2),\ E,\ (1,1)) + 0.95V^{t}(1,1)]$$
$$= 0.5\,[-\ 0.1 + 0.95(-\ 0.00)]$$
$$+ 0.5\,[-\ 0.1 + 0.95(0.082)]$$
$$=-\ 0.06105$$

$$Q((1,\ 2),\ W)$$
$$= P((1,1) \mid (1,\ 2),\ W)\,[R((1,2),\ W,\ (1,1)) + 0.95V^{t}(1,1)]$$
$$+ P((1,2) \mid (1,\ 2),\ W)\,[R((1,2),\ W,\ (1,2)) + 0.95V^{t}(1,2)]$$
$$= 0.5\,[-\ 0.1 + 0.95(0.082)]$$
$$+ 0.5\,[-\ 0.1 + 0.95(-\ 0.11)]$$
$$=-\ 0.1133$$

$$Q((1,\ 2),\ S)$$
$$= P((1,2) \mid (1,\ 2),\ S)\,[R((1,2),\ S,\ (1,2)) + 0.95V^{t}(1,2)]$$
$$+ P((1,3) \mid (1,\ 2),\ S)\,[R((1,2),\ S,\ (1,3)) + 0.95V^{t}(1,3)]$$
$$+ P((1,1) \mid (1,\ 2),\ S)\,[R((1,2),\ S,\ (1,1)) + 0.95V^{t}(1,1)]$$
$$= 0.5\,[-\ 0.1 + 0.95(-\ 0.11)]$$
$$+ 0.25\,[-\ 0.1 + 0.95(-\ 0.00)]$$
$$+ 0.25\,[-\ 0.1 + 0.95(0.082)]$$
$$=-\ 0.132775$$

$$V'(s) = max\ Q'(s,a)$$

$$V^{t+1}(1,\ 2) =-\ 0.06105$$

$$
\begin{aligned}
&Q((1,\ 3),\ N)\\
&= P((2,3)\mid(1,\ 3),\ N)\,[R((1,3),\ N,\ (2,3))\ +\ 0.95V^{t}(2,3)]\\
&+\ P((1,4)\mid(1,\ 3),\ N)\,[R((1,3),\ N,\ (1,4))\ +\ 0.95V^{t}(1,4)]\\
&+\ P((1,2)\mid(1,\ 3),\ N)\,[R((1,3),\ N,\ (1,2))\ +\ 0.95V^{t}(1,2)]\\
&=\ 0.5\,[-\ 0.1\ +\ 0.95(0.434)]\\
&+\ 0.25\,[-\ 0.1\ +\ 0.95(-\ 0.35)]\\
&+\ 0.25\,[-\ 0.1\ +\ 0.95(-\ 0.11)]\\
&=-\ 0.0031
\end{aligned}
$$

$$
\begin{aligned}
&Q((1,\ 3),\ E)\\
&= P((1,4)\mid(1,\ 3),\ E)\,[R((1,3),\ E,\ (1,4))\ +\ 0.95V^{t}(1,4)]\\
&+\ P((2,3)\mid(1,\ 3),\ E)\,[R((1,3),\ E,\ (2,3))\ +\ 0.95V^{t}(2,3)]\\
&+\ P((1,3)\mid(1,\ 3),\ E)\,[R((1,3),\ E,\ (1,3))\ +\ 0.95V^{t}(1,3)]\\
&=\ 0.5\,[-\ 0.1\ +\ 0.95(-\ 0.35)]\\
&+\ 0.25\,[-\ 0.1\ +\ 0.95(0.434)]\\
&+\ 0.25\,[-\ 0.1\ +\ 0.95(-\ 0.00)]\\
&=-\ 0.163175
\end{aligned}
$$

$$
\begin{aligned}
&Q((1,\ 3),\ W)\\
&= P((1,2)\mid(1,\ 3),\ W)\,[R((1,3),\ W,\ (1,2))\ +\ 0.95V^{t}(1,2)]\\
&+\ P((2,3)\mid(1,\ 3),\ W)\,[R((1,3),\ W,\ (2,3))\ +\ 0.95V^{t}(2,3)]\\
&+\ P((1,3)\mid(1,\ 3),\ W)\,[R((1,3),\ W,\ (1,3))\ +\ 0.95V^{t}(1,3)]\\
&=\ 0.5\,[-\ 0.1\ +\ 0.95(-\ 0.11)]\\
&+\ 0.25\,[-\ 0.1\ +\ 0.95(0.434)]\\
&+\ 0.25\,[-\ 0.1\ +\ 0.95(-\ 0.00)]\\
&=-\ 0.049175
\end{aligned}
$$

$$
\begin{aligned}
&Q((1,\ 3),\ S)\\
&= P((1,3)\mid(1,\ 3),\ S)\,[R((1,3),\ S,\ (1,3))\ +\ 0.95V^{t}(1,3)]\\
&+\ P((1,4)\mid(1,\ 3),\ S)\,[R((1,3),\ S,\ (1,4))\ +\ 0.95V^{t}(1,4)]\\
&+\ P((1,2)\mid(1,\ 3),\ S)\,[R((1,3),\ S,\ (1,2))\ +\ 0.95V^{t}(1,2)]\\
&=\ 0.5\,[-\ 0.1\ +\ 0.95(-\ 0.00)]\\
&+\ 0.25\,[-\ 0.1\ +\ 0.95(-\ 0.35)]\\
&+\ 0.25\,[-\ 0.1\ +\ 0.95(-\ 0.11)]\\
&=-\ 0.20925
\end{aligned}
$$

$$V^{t+1}(1,\ 3)\ =-\ 0.0031$$

$$
\begin{aligned}
&Q((1,\ 4),\ N)\\
&= P((2,4)\mid(1,\ 4),\ N)\,[R((1,4),\ N,\ (2,4))\ +\ 0.95V^{t}(2,4)]\\
&+\ P((1,4)\mid(1,\ 4),\ N)\,[R((1,4),\ N,\ (1,4))\ +\ 0.95V^{t}(1,4)]\\
&+\ P((1,3)\mid(1,\ 4),\ N)\,[R((1,4),\ N,\ (1,3))\ +\ 0.95V^{t}(1,3)]
\end{aligned}
$$

$= 0.5\,[-\ 1\ + 0.95(-\ 1)]$
$+\ 0.25\,[-\ 0.1\ + 0.95(-\ 0.35)]$
$+\ 0.25\,[-\ 0.1\ + 0.95(-\ 0.00)]$
$=-\ 1.108125$

$Q((1,\ 4),\ E)$
$= P((1,4)\mid(1,\ 4),\ E)\,[R((1,4),\ E,\ (1,4))\ + 0.95V^{t}(1,4)]$
$+\ P((2,4)\mid(1,\ 4),\ E)\,[R((1,4),\ E,\ (2,4))\ + 0.95V^{t}(2,4)]$
$= 0.75\,[-\ 0.1\ + 0.95(-\ 0.35)]$
$+\ 0.25\,[-\ 1\ + 0.95(-\ 1)]$
$=-\ 0.811875$

$Q((1,\ 4),\ W)$
$= P((1,3)\mid(1,\ 4),\ W)\,[R((1,4),\ W,\ (1,3))\ + 0.95V^{t}(1,3)]$
$+\ P((2,4)\mid(1,\ 4),\ W)\,[R((1,4),\ W,\ (2,4))\ + 0.95V^{t}(2,4)]$
$+\ P((1,4)\mid(1,\ 4),\ W)\,[R((1,4),\ W,\ (1,4))\ + 0.95V^{t}(1,4)]$
$= 0.5\,[-\ 0.1\ + 0.95(-\ 0.00)]$
$+\ 0.25\,[-\ 1\ + 0.95(-\ 1)]$
$+\ 0.25\,[-\ 0.1\ + 0.95(-\ 0.35)]$
$=-\ 0.645625$

$Q((1,\ 4),\ S)$
$= P((1,4)\mid(1,\ 4),\ S)\,[R((1,4),\ S,\ (1,4))\ + 0.95V^{t}(1,4)]$
$+\ P((1,3)\mid(1,\ 4),\ S)\,[R((1,4),\ S,\ (1,3))\ + 0.95V^{t}(1,3)]$
$= 0.75\,[-\ 0.1\ + 0.95(-\ 0.35)]$
$+\ 0.25\,[-\ 0.1\ + 0.95(-\ 0.00)]$
$=-\ 0.349375$

$V^{t+1}(1,\ 4)\ =-\ 0.349375$

V^{t+1} for different states at iteration t+1

3	0.653	1.059	1.381	**Food**
2	0.400	blocked	0.434	**Tiger**
1	0.082	-0.06105	-0.0031	-0.349375
	1	**2**	**3**	**4**

(b) What is the policy for each state as per the Q- function $Q^{t+1}(..)$. Note it down below for each entry with "?" mark.

3	E	E	E	**Food**
2	N	blocked	N	**Tiger**
1	N	**?**	**?**	**?**
	1	**2**	**3**	**4**

Please provide clear **analytical derivations** for extracting the policy as per Q^{t+1} (**without** using any python code).

$$\pi^*(s) = arg\,max\, Q^{t+1}(s, a)$$

$$\pi^*(1, 2) = E$$
$$\pi^*(1, 3) = N$$
$$\pi^*(1, 4) = S$$

policy for each state

3	E	E	E	Food
2	N	blocked	N	Tiger
1	N	E	N	S
	1	**2**	**3**	**4**

Question 4 [10 Points]

In this question, you will employ Singular Value Decomposition to obtain word embeddings and compare the generated word embeddings with the word embeddings generated using word2vec. The corpus (or dataset) to be considered is a set of tweets posted about the Covid-19 pandemic on Twitter (Corona_Tweets.csv), which is a real-life dataset. You need to do the following (code template provided is **Q4_template_tweets.ipynb**; you are free to use helper functions if required):

(a) Update the load_data function from the Q4_template_tweets.ipynb to preprocess words: remove non-letters, convert words into the lower case, and remove the stop words. You can employ some functions from the "NLP-pipeline-example.ipynb" example and may use regular expressions from "Word2Vec.ipynb". **[2 marks]**

```
stopWords = set(stopwords.words('english'))

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('Corona_Tweets.csv', names=['text'])
    list_words = []
    ### iterate over all tweets from the dataset
    for i in tweets.index:
      ### remove non-letter.
      text = re.sub("[^a-zA-Z]"," ", tweets['text'][i])

      ### tokenize
      words = nltk.word_tokenize(text)

      new_words = []
      ### iterate over all words of a tweet
      for w in words:
        ## TODO: remove the stop words and convert a word (w) to the lower case
        if w not in stopWords:
          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])
```

Output:

```
[['trending', 'new', 'yorkers', 'encounter', 'empty', 'supermarket', 'shelves',
'pictured', 'wegmans', 'brooklyn', 'sold', 'online', 'grocers', 'foodkick',
'maxdelivery', 'coronavirus', 'fearing', 'shoppers', 'stock', 'https', 'co', 'gr',
'pcrlwh', 'https', 'co', 'ivmkmsqdt'], ['when', 'i', 'find', 'hand', 'sanitizer',
'fred', 'meyer', 'i', 'turned', 'amazon', 'but', 'pack', 'purell', 'check',
'coronavirus', 'concerns', 'driving', 'prices', 'https', 'co', 'ygbipbflmy'], ['find',
'protect', 'loved', 'ones', 'coronavirus']]
```

(b) Create the co-occurrence matrix for all the remaining words (after stop words are eliminated), where the window of co-occurrence is 5 on either side of the word. What is the size of your vocabulary (i.e., how many unique words you end up with)? **[4 marks]**

```
def distinct_words(corpus):
    """ get a list of distinct words for the corpus.
        Params:
            corpus (list of list of strings): corpus of documents
        Return:
            corpus_words (list of strings): list of distinct words across the corpus,
sorted (using python 'sorted' function)
            num_corpus_words (integer): number of distinct words across the corpus
    """
    corpus_words = []

    # ------------------
    # TODO:
    # ------------------

    corpus = [w for x in corpus for w in x]
    corpus_words = list(set(corpus))
    corpus_words = sorted(corpus_words)

    return corpus_words, len(corpus_words)

words, num_words = distinct_words(twitter_corpus)
print(words[:10], num_words)
```

Output:

```
['a', 'aadya', 'aadyasitara', 'aah', 'aamiin', 'aapl', 'aaq', 'ab', 'abajam', 'abandon']
13967
```

There are 13967 distinct words.

```
def compute_co_occurrence_matrix(corpus, window_size=5):
    """ Compute co-occurrence matrix for the given corpus and window_size (default of 5).
        Params:
            corpus (list of list of strings): corpus of documents
            window_size (int): size of context window
        Return:
```

```
            M (numpy matrix of shape = [number of corpus words x number of corpus
words]):
                Co-occurence matrix of word counts.
                The ordering of the words in the rows/columns should be the same as the
ordering of the words given by the distinct_words function.
            word2Ind (dict): dictionary that maps word to index (i.e. row/column number)
for matrix M.
    """
    M = None
    word2Ind = {}

    # ------------------
    # TODO:
    # ------------------
    words, num_words = distinct_words(corpus)
    print("Vocabulary size:", num_words)

    M = np.zeros((num_words, num_words))
    for i in range(num_words):
        word2Ind[words[i]] = i

    for sentence in corpus:
        for i in range(len(sentence)):
            ci = word2Ind[sentence[i]]

            for l in sentence[max(0, i - window_size) : i]:
                wi = word2Ind[l]
                M[ci][wi] += 1

            for r in sentence[i + 1 : i + 1 + window_size]:
                wi = word2Ind[r]
                M[ci][wi] += 1

    return M, word2Ind

M, word2Ind = compute_co_occurrence_matrix(twitter_corpus)
print(M)
```

Output:

```
Vocabulary size: 13967
[[4. 0. 0. ... 0. 0. 0.]
 [0. 0. 1. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

(c) Apply SVD and obtain word embeddings of size 75. **[2 marks]**

```
# ----------------------------
# Run SVD
# Note: This may take several minutes (~20-30 minutes)
```

```
# ------------------------------
size = 75

import numpy as np
U, s, Vh = np.linalg.svd(M, full_matrices=False)
S = np.diag(s)  # Constructing a diagonal matrix from the singular values
SVD_embeddings = np.dot(U[:, :size], np.dot(S[:size, :size], Vh[:size, :])) #
Reconstructing the matrix using the left singular vectors, singular values, and right
singular vectors
print(SVD_embeddings)
```

Output:

```
[[ 2.77876631e+00 -4.57275844e-02 -4.33800790e-02 ...  9.81315087e-03
   1.89509765e-02 -2.67398659e-03]
 [-4.57275844e-02  1.12165694e-02  1.22207172e-02 ...  6.72758601e-04
  -3.60285631e-04 -5.67004159e-04]
 [-4.33800790e-02  1.22207172e-02  1.25810419e-02 ... -1.01807675e-04
  -2.95448614e-04  2.40141078e-04]
 ...
 [ 9.81315087e-03  6.72758601e-04 -1.01807675e-04 ... -1.19126760e-04
   5.77708145e-04  2.25316256e-03]
 [ 1.89509765e-02 -3.60285631e-04 -2.95448614e-04 ...  5.77708145e-04
   4.66652504e-04  1.21542440e-03]
 [-2.67398659e-03 -5.67004159e-04  2.40141078e-04 ...  2.25316256e-03
   1.21542440e-03 -1.76263959e-04]]
```

(d) Then, please generate word embeddings of size 75 using Word2Vec.pynb (uploaded in class lecture material) on the same dataset. Please show comparison on few examples to understand which method works better. You may use the svd.most_similar function from the template to perform the comparisons.

Note your observations in your solution. You can use words like “covid”, “grocery” etc to compare the two models. **[2 marks]**

Show your code snippets that you used for different sub-parts in your pdf.

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

# Initializing the train model.
num_features = 100    # Word vector dimensionality
min_word_count = 50   # Minimum word count. You can change it also.
num_workers = 4      # Number of parallel threads, can be changed
context = 10          # Context window size
downsampling = 1e-3 # (0.001) Downsample setting for frequent words, can be changed
# Initializing the train model
print("Training Word2Vec model....")
model = word2vec.Word2Vec(twitter_corpus,\
                          workers=num_workers,\
                          size=num_features,\
                          min_count=min_word_count,\
                          window=context,
                          sample=downsampling)
```

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

```
from sklearn.metrics.pairwise import cosine_similarity

def svd_most_similar(query_word, n=10):
   """ return 'n' most similar words of a query word using the SVD word embeddings
similar to word2vec's most_smilar
       Params:
           query_word (strings): a query word
       Return:
           most_similar (list of strings): the list of 'n' most similar words
   """
   # get index of a query_word
   query_word_idx = word2Ind[query_word]
   # get word embedding for a query_word
   word = SVD_embeddings[query_word_idx]
   #cosine similarity matrix
   cos_similarity = cosine_similarity(SVD_embeddings, word.reshape(1, -1))
   most_similar = []

   # Write additional code to compute the list most_similar. Each entry in the list is a
tuple (w, cos)
   # where w is one of the most similar word to query_word and cos is cosine similarity
of w with query_word

   # get indices of top n most similar words (excluding the query word itself)
   top_n_indices = np.argsort(cos_similarity.flatten())[:-n-1:-1][1:]

   # get the words corresponding to the top n indices
   for idx in top_n_indices:
       most_similar.append((list(word2Ind.keys())[list(word2Ind.values()).index(idx)],
cos_similarity[idx][0]))

   return most_similar
```

```
svd_most_similar("covid")
```

Output:

```
[('emptyshelves', 0.9232596240136255),
 ('toiletpaper', 0.9158881719437213),
 ('w', 0.9078679839083614),
 ('amid', 0.8977997425692742),
 ('staythefhome', 0.8933014456365583),
 ('read', 0.8867900093748593),
 ('overflowing', 0.8867885168732753),
 ('stoppanicbuying', 0.8830587385520966),
 ('report', 0.8826697156427498)]
```

```
model.wv.most_similar("covid") #this word2vec trained model on tweets
```

Output:

```
[('coronavirus', 0.9999841451644897),
 ('e', 0.9999696612358093),
 ('coronavirusoutbreak', 0.9999670386314392),
 ('x', 0.9999657869338989),
 ('n', 0.9999630451202393),
 ('c', 0.9999605417251587),
 ('panicbuying', 0.9999600052833557),
 ('b', 0.9999579787254333),
 ('coronaoutbreak', 0.9999568462371826),
 ('p', 0.9999552369117737)]
```

As Word2Vec calculates similarity between word vectors using cosine similarity, it may not capture words with a meaningful semantic relationship with “covid”, but rather words that are so common in the corpus, they may be highly associated with many other words, including "covid".

```
svd_most_similar("grocery")
```

Output:

```
[('mailing', 0.8930877074465083),
 ('liquor', 0.88683832488559),
 ('mall', 0.8850624401278997),
 ('accusations', 0.8767469983372158),
 ('elys', 0.8761281369249165),
 ('llama', 0.8752093202749379),
 ('pajama', 0.8737340360766817),
 ('dollargeneral', 0.8732091037749703),
 ('quarterly', 0.8722804891789773)]
```

```
model.wv.most_similar("grocery")
```

Output:

```
[('local', 0.9998742341995239),
 ('today', 0.9998664855957031),
 ('shelves', 0.999864399433136),
 ('i', 0.9998546838760376),
 ('my', 0.9998495578765869),
 ('every', 0.9998405575752258),
 ('going', 0.9998399615287781),
 ('back', 0.999835729598999),
 ('line', 0.9998341798782349),
 ('things', 0.9998293519020081)]
```

When SVD is used to generate word embeddings, the embeddings are determined by the patterns of word co-occurrences in the corpus, rather than by any explicit modelling of word meaning. Hence, the words that are outputted as most similar to “grocery” may be similar in the sense that they may co-occur in similar contexts, but this similarity may not have any meaningful semantic or conceptual connection.

Question 5 [10 Points]

In this question, you need to install OpenAI gym:

- https://github.com/openai/gym (installation instructions)
- https://www.gymlibrary.dev/content/basic_usage/ (documentation)

Once the gym is installed, you have to implement Q-learning for the **FrozenLake-v1** environment (https://www.gymlibrary.dev/environments/toy_text/frozen_lake/) in python using the gym library and show the rewards obtained. Use option **is_slippery=True**. Use map size 4x4 (check option `map_name').

You may use a discount factor of 0.95. Please provide the following things in your solution for this question:

(a) Code that implements Q-learning for the FrozenLake-v1 example in the gym? Copy and paste the code in the solution pdf, and provide the actual code file also. A key thing to determine first is what is the size of the state space and action space in this environment to make the appropriate data structures such as the Q-table [**6 points]**

```
import numpy as np
import random
import gym
env = gym.make('FrozenLake-v1', desc=None, map_name="4x4", is_slippery=True)

# Getting the state space
print("Action Space {}".format(env.action_space))
print("State Space {}".format(env.observation_space))

# Setting the hyperparameters
alpha = 0.1 #learning rate
discount_factor = 0.95
epsilon = 0.1
train_episodes = 100000
test_episodes = 50

# Initialising the Q-table
Q = np.zeros((env.observation_space.n, env.action_space.n))
print(Q)
```

Output:

```
Action Space Discrete(4)
State Space Discrete(16)
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
```

```
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

The size of the state space is 16, as the grid is of size 4x4. The size of the action space is 4 as the agent can move in 4 directions (up, down, left, right). These sizes were used to create the Q table of size 16x4.

```
# Reward table
# {action: [(probability, nextstate, reward, done)]}
env.P[14]
```

Output:

```
{0: [(0.3333333333333333, 10, 0.0, False),
  (0.3333333333333333, 13, 0.0, False),
  (0.3333333333333333, 14, 0.0, False)],
 1: [(0.3333333333333333, 13, 0.0, False),
  (0.3333333333333333, 14, 0.0, False),
  (0.3333333333333333, 15, 1.0, True)],
 2: [(0.3333333333333333, 14, 0.0, False),
  (0.3333333333333333, 15, 1.0, True),
  (0.3333333333333333, 10, 0.0, False)],
 3: [(0.3333333333333333, 15, 1.0, True),
  (0.3333333333333333, 10, 0.0, False),
  (0.3333333333333333, 13, 0.0, False)]}
```

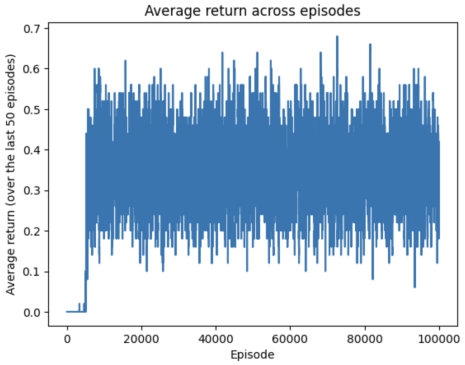
```
#Training the agent
train_reward_list = []

for episode in range(train_episodes):
   episode_reward = 0

   #Resetting the environment each time as per requirement
   state = env.reset()
   #Starting the tracker for the rewards
   done = False
    while True:
     ### STEP 1: FIRST option for choosing the initial action - exploit
     #If the random number is larger than epsilon: employing exploitation
     #and selecting best action
     if random.uniform(0, 1)  > epsilon:
         action = np.argmax(Q[state])

     ### STEP 1: SECOND option for choosing the initial action - explore
     #Otherwise, employing exploration: choosing a random action
     else:
```

```
            action = env.action_space.sample()

        ### STEP 2: performing the action and getting the reward
        next_state, reward, done, info = env.step(action)
        episode_reward += reward

        ### STEP 3: update the Q-table
        # Updating the Q-table using the Bellman equation
        Q[state, action] = (1 - alpha) * Q[state, action] + alpha * (reward +
discount_factor * np.max(Q[next_state]))
        # Increasing our total reward and updating the state
        state = next_state

        # Ending the episode
        if done:
          break

    train_reward_list.append(episode_reward)
```

(b) For each episode, compute the total accumulated reward (also called episode return). Plot the average return (over the last 50 episodes) while your agent is learning (x-axis will be the episode number, y-axis will be the average return over the last 50 episodes). Make sure that you train for sufficiently many episodes so that convergence occurs. **[3 points]**

```
avg_reward = []
for i in range(50, train_episodes):
 avg_reward.append(np.mean(train_reward_list[i-50:i]))

x = range(50, train_episodes)
plt.plot(x, avg_reward)
plt.xlabel('Episode')
plt.ylabel('Training total reward')
plt.title('Average rewards')
plt.show()
```

Output:




```
# Testing out the model
test_reward_list = []
steps_list = []
misses = 0
```

```
for episode in range(test_episodes):
    steps = 0
    episode_reward = 0
    state = env.reset()
    done = False

    while True:
        # Choose the action with the highest Q-value
        action = np.argmax(Q[state, :])

        # Take the action and observe the next state and reward
        next_state, reward, done, _ = env.step(action)
        steps += 1

        # Add the reward to the total
        episode_reward += reward

        # Move to the next state
        state = next_state

        if done and reward == 1:
          # print('You reached the goal after {} steps'.format(steps))
          steps_list.append(steps)
          break
        elif done and reward == 0:
          # print("You fell in a hole!")
          misses += 1
          break

    test_reward_list.append(episode_reward)

print('-----------------------------------------------')
print('You took an average of {:.0f} steps to reach the
goal'.format(np.mean(steps_list)))
print('And you fell in the hole {:.2f} % of the times'.format((misses/test_episodes) *
100))
print('-----------------------------------------------')
```

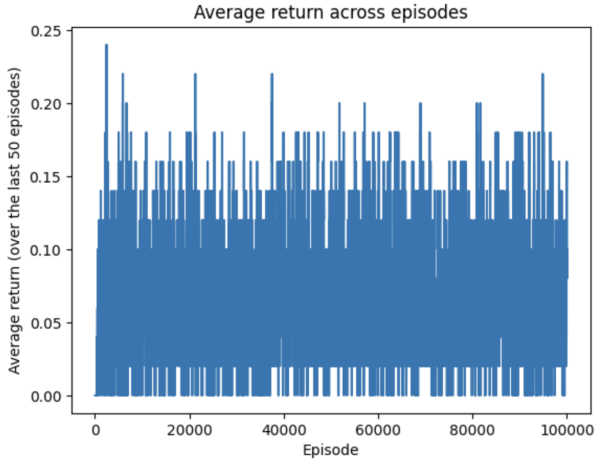
Output:

```
-----------------------------------------------
You took an average of 36 steps to reach the goal
And you fell in the hole 24.00 % of the times
-----------------------------------------------
```

(c) What is the epsilon you used for epsilon-greedy exploration and how did you select this epsilon? What would happen if you use a high epsilon such as 0.5? **[1 point]**

The goals of this question are to ensure you familiarize yourself with OpenAI Gym and how to implement Q-learning in OpenAI Gym. There are many resources available online on implementing Q-learning in Gym. You are free to refer to them, but please write your own code. Please use the standard tabular Q-learning for this question (without using any neural nets or any other function approximator).

I used epsilon = 0.1 because it is a common value used for epsilon-greedy exploration. If a high epsilon value such as 0.5 is used, then the agent will be more likely to choose a random action instead of exploiting the current knowledge of the Q-values. This means that the agent will explore the environment more, which can be useful in the beginning of the learning process when the Q-values are not well-defined. However, as the agent continues to learn, it may become beneficial to reduce the exploration rate and increase the exploitation rate so that the agent can use its knowledge of the Q-values to make more informed decisions. If the exploration rate is too high throughout the entire learning process, the agent may not converge to an optimal policy and may instead continue to make suboptimal decisions.



Alternatively, a high epsilon value can be used as a decaying epsilon that starts with a high exploration rate and gradually reduces it as the agent learns. For example, you can set a high value for epsilon at the beginning of training and gradually decrease it over time. This approach allows the agent to explore the environment more at the beginning and focus more on exploiting the learned policy as it becomes more experienced.

References:

1. https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/
2. https://medium.com/swlh/introduction-to-q-learning-with-openai-gym-2d794da10f3d
3. https://medium.com/analytics-vidhya/solving-the-frozenlake-environment-from-openai-gym-using-value-iteration-5a078dffe438