A
LAB REPORT
ON
Artificial Intelligence In Education

By

Rajesh Kumar Yadav

Exam Roll No: 7629020

Registration No.9-2-29-630-2019

Submitted to:

Arun Sharma

Mahendra Ratna Campus

In partial fulfillment of the requirements for the Course
Artificial Intelligence In Education

Tahachal, Kathmandu

April,2024

Table Of Content

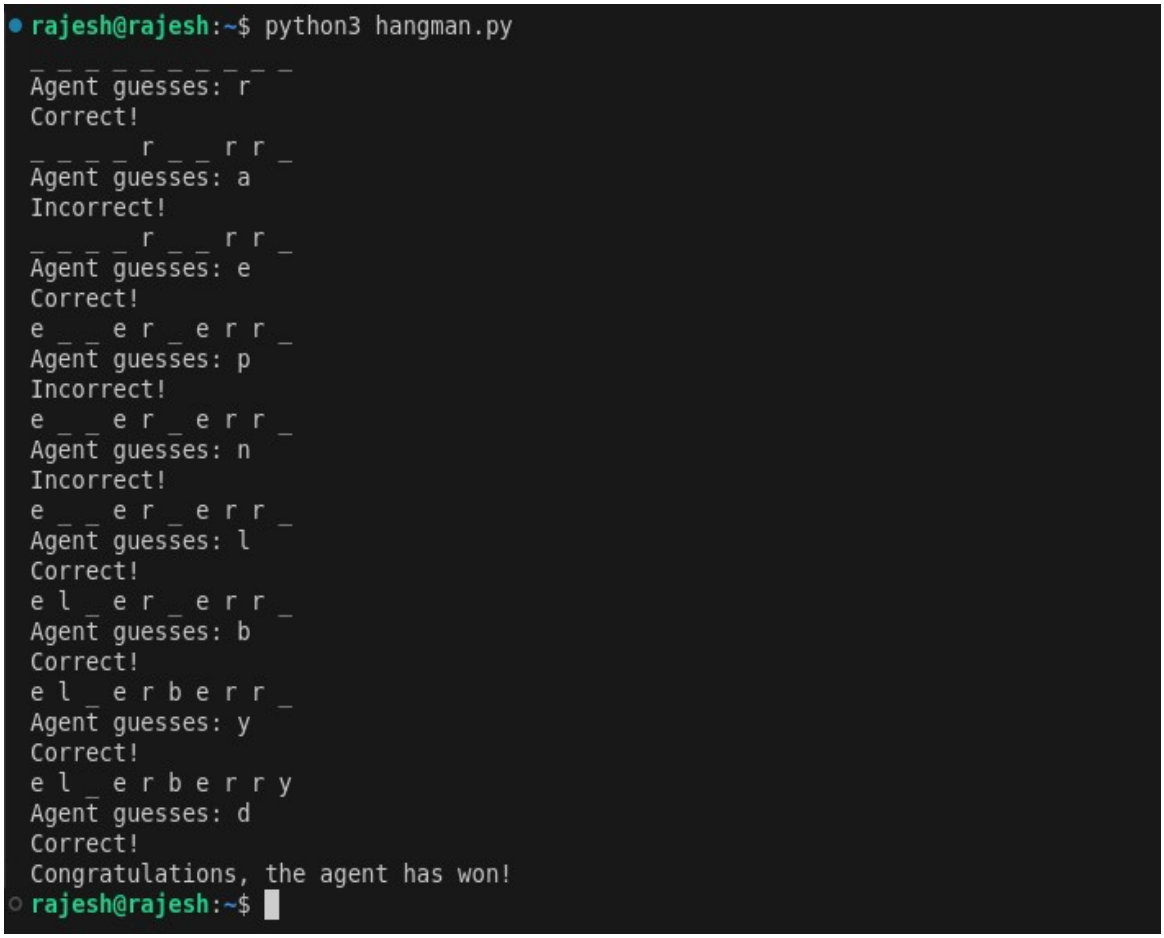
1 Write programs for Hangman (implementation of simple intelligent agent)

1.1 Source code

```
import random



words=[‘apple`,’banana’,’cherry’,’durian’,’elderberry’,’fig’,’grape’]


word =random.choice(words)



word_progress=[‘_’ for i in range (len(word))]


max_guesses=6


incorrect_guesses=[]


letter_frequency={}
for w in words:
for letter in w:
if letter in letter_frequency:
letter_frequency[letter]+=1


else:
letter_frequency[letter]=1


sorted_letters=sorted(letter_frequency,key=letter_frequency.get,reverse=True)


def intelligent_guess(word_progress,incorrect_guesses,sorted_letters):
for letter in sorted_letters:
if letter not in word_progress and letter not in incorrect_guesses:
return letter
return None

while True:

print(‘’.join(word_progress))

guess=intelligent_guess(word_progress,incorrect_guesses,sorted_letters)
print(f”Agent guesses:{guess}”)


if guess in incorrect_guesses or guess in word_progress:
print(“Agent guessed a letter that was already guessed!”)
continue

if guess in word:

for i in range(len(word)):

if word[i]== guess:
word_progress[i]=guess
print(“Correct!”)
else:
```

```
incorrect_guesses.append(guess)
print(“Incorrect”)

if’_’not in word_progress:
print(“Congratulations,the agent has won!”)
break

if len(incorrect_guesses)>=max_guesses:
print(“Sorry,the agent has lost. The word was ’{word}’.”)
break
```

1.2 Output Window



2 Write programs for BFS from queue import Queue

2.1 Source code

From queue import Queue

```
graph={
‘A’:[‘B’,’C’],
‘B’:[‘D’,’E’],
‘C’:[‘F’],
‘D’:[],
‘E’:[‘F’],
‘F’:[]
}

def bfs(graph,start):

visited=set()

queue=Queue()

queue.put(start)
```

while not queue.empty():

node=queue.get()
if node not in visited:

print(node)

visited.add(node)

for neighbor in graph[node]:
if neighbor not in visited:
queue.put(neighbor)

bfs(graph,’A’)

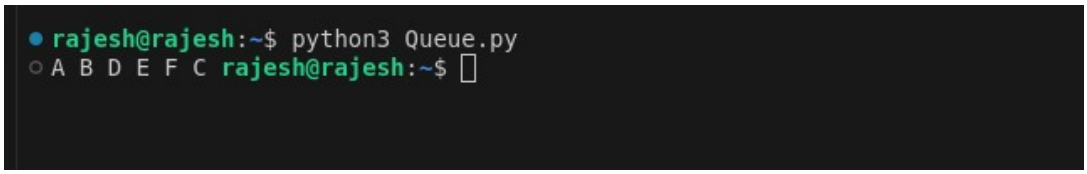
2.2 Output Window

3 Write programs for DFS.

3.1 Source code

```
graph={
'A':['B','C'],
'B':['D','E'],
'C':['F'],
'D':[],
'E':['F'],
'F':[]
}

def dfs(graph,start,visited=None):

if visited is None:
visited=set()

visited.add(start)
print(start,end='')

  recursively
for neighbor in graph[start]:
if neighbour not in visited:
dfs(graph,neighbour,visited)


dfs(graph,'A')
```

3.2 Output Window



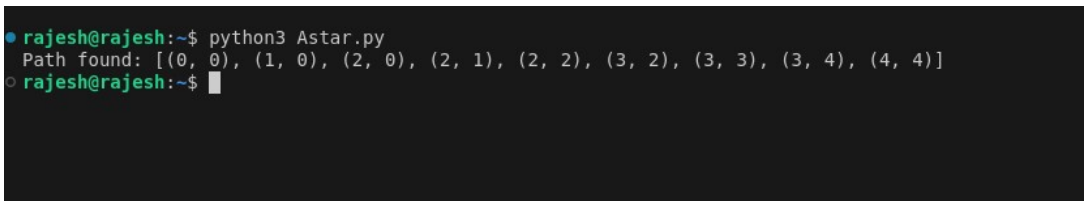
4 Write a program for A*

4.1 Source code

```
Import heapq
grid=[
[0,1,0,0,0],
[0,1,0,1,0],
[0,0,0,1,0],
[1,1,0,0,0],
[0,0,0,1,0]
]

start=(0,0)
goal(4,4)
def heuristic(a,b):
return abs(a[0]-v[0]+abs(a[1]-b[1])

def astar(grid,start,goal):
rows,cols=len(grid),len(grid[0])
open_set=[]
```

```
heapq.heappush(open_set,(0,start))
came_from={}
g_score={start:0}
f_score={start:heuristic(start,goal)}

while open_set:
current=heapq.heappop(open_set)[1]

if current==goal:
path=[]
while current in came from:
path.append(current)
current=came_from[current]
path.reverse()
return path

neighbors=[(current[0]+i,current[1]+j) for I,j in [(-1,0),(1,0),(0,-1),(0,1)]]
valid_neighbors=[n for n in neighbors if 0 <=n[0]<rows and 0 <=n[1]<cols and grid[n[0]]
[n[1]]==0]

for neighbor in valid_neighbors:
tentative_g_score=g_score[current]+1
if neighbor not in g_score or tentative_g_score<g_score[neighbor]:
came_from[neighbor]=current
g_score[neighbor]=tentative_g_score

f_score[neighbor]=tentative_g_score+heuristic(neighbor,goal)
heapq.heappush(open_set,(f_score[neighbor],neighbor))
return None

path=astar(grid,start,goal)
if path:
print(“Path found:”,path)
else:
print(“No path found”)
```

4.2 Output Window



5 Write programs for Greedy Best First

5.1 Source code

```
InputGraph={
‘A’:[(‘B’,4),(‘D’,12)],
‘B’:[(‘A’,9),(‘C’,4)],
‘C’:[(‘D’,4),(‘B’,7)],
‘D’:[(‘A’,9),(‘C’,5)]
}

goal=”C”

def gbfs(graph,start):

queue=[(start,0)]
visitedNodes=[]

while queue:
queue=sorted(queue,key=lamda x:x[1])

node,heuristic=queue.pop(0)
if node not in visitedNodes:
visitedNodes.append(node)
if node==goal:
break
neighbors =graph[node]
for neighbor in neighbors:
queue.append(neighbor)
return visitedNodes
print(gbfs(inputGraph,”A”))
```

5.2 Output Window