Vaibhav Sharma
200365101

Computer Science 320
Introduction to Artificial Intelligence

Department of Computer Science
University of Regina
Fall 2019

Assignment 2

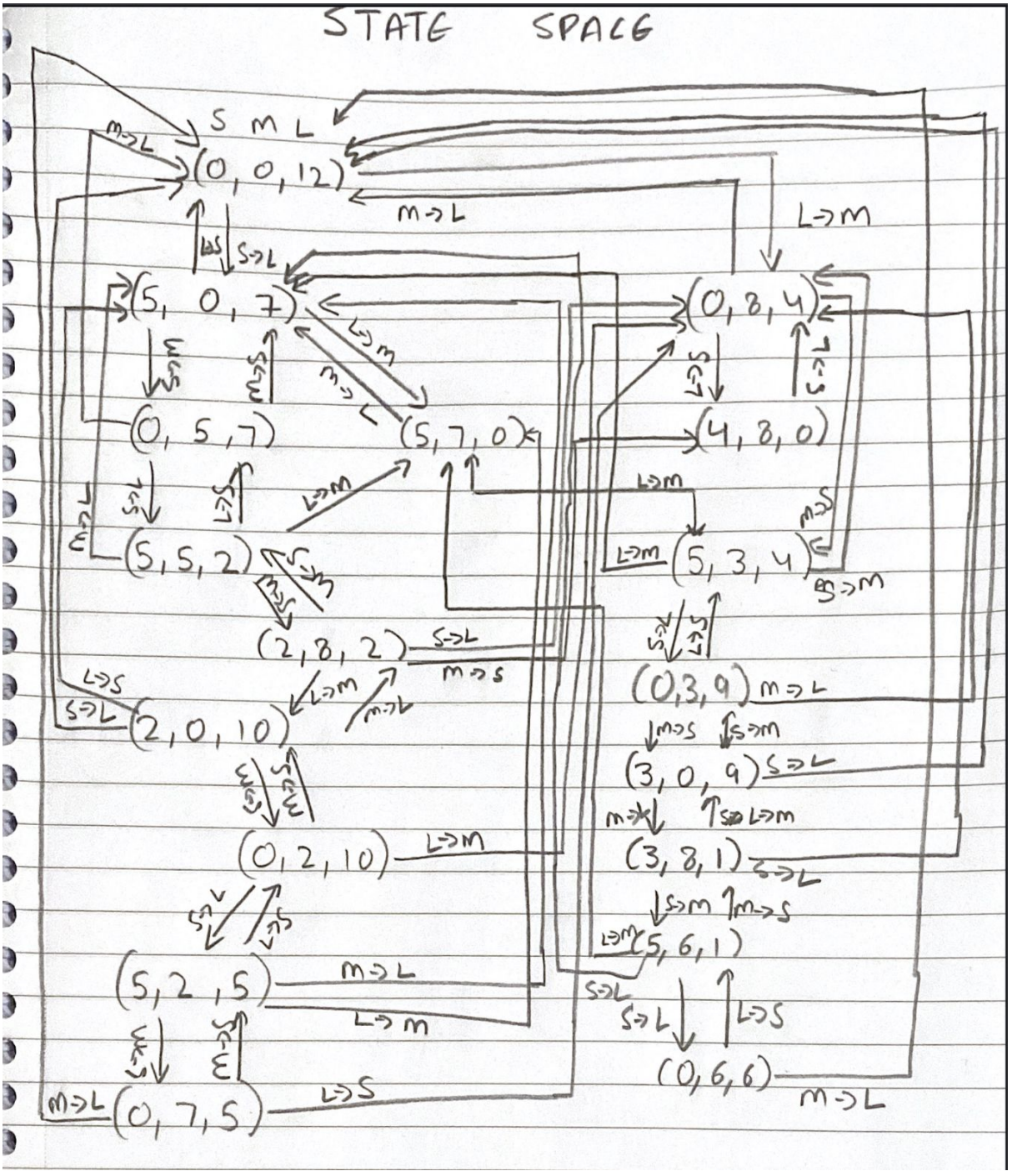
Handout date: September 30, 2019
Due Date: October 16, 2019

1. (10 marks)
Consider the following problem:

Rowena has three unmarked glasses of different sizes:
5 ounces, 8 ounces, and 12 ounces.
The largest glass is full.
What can Rowena do to get 6 ounces of liquid into
each of the larger two glasses?

Use state-space-search method to solve this problem. Draw EXPLICITLY the state space and give the solution based on both DFS and BFS.

Solution ->



Using DFS -> Shortest path to (0,0,6)

1. (0,0,12)
2. (5,0,7)
3. (0,5,7)
4. (5,5,2)
5. (2,8,2)

6. (0,8,4)
7. (5,3,4)
8. (0,3,9)
9. (3,0,9)
10. (3,8,1)
11. (5,6,1)
12. (0,6,6)

Using BFS -> Shortest path to (0,0,6)

1. (0,0,12)
2. (5,0,7)
3. (0,8,4)
4. (0,5,7)
5. (5,7,0)
6. (4,8,0)
7. (5,3,4)
8. (5,5,2)
9. (0,3,9)
10. (2,8,2)
11. (3,0,9)
12. (2,0,10)
13. (3,8,1)
14. (5,2,5)
15. (5,6,1)
16. (0,6,6)

2. (10 marks)

Consider the following story:

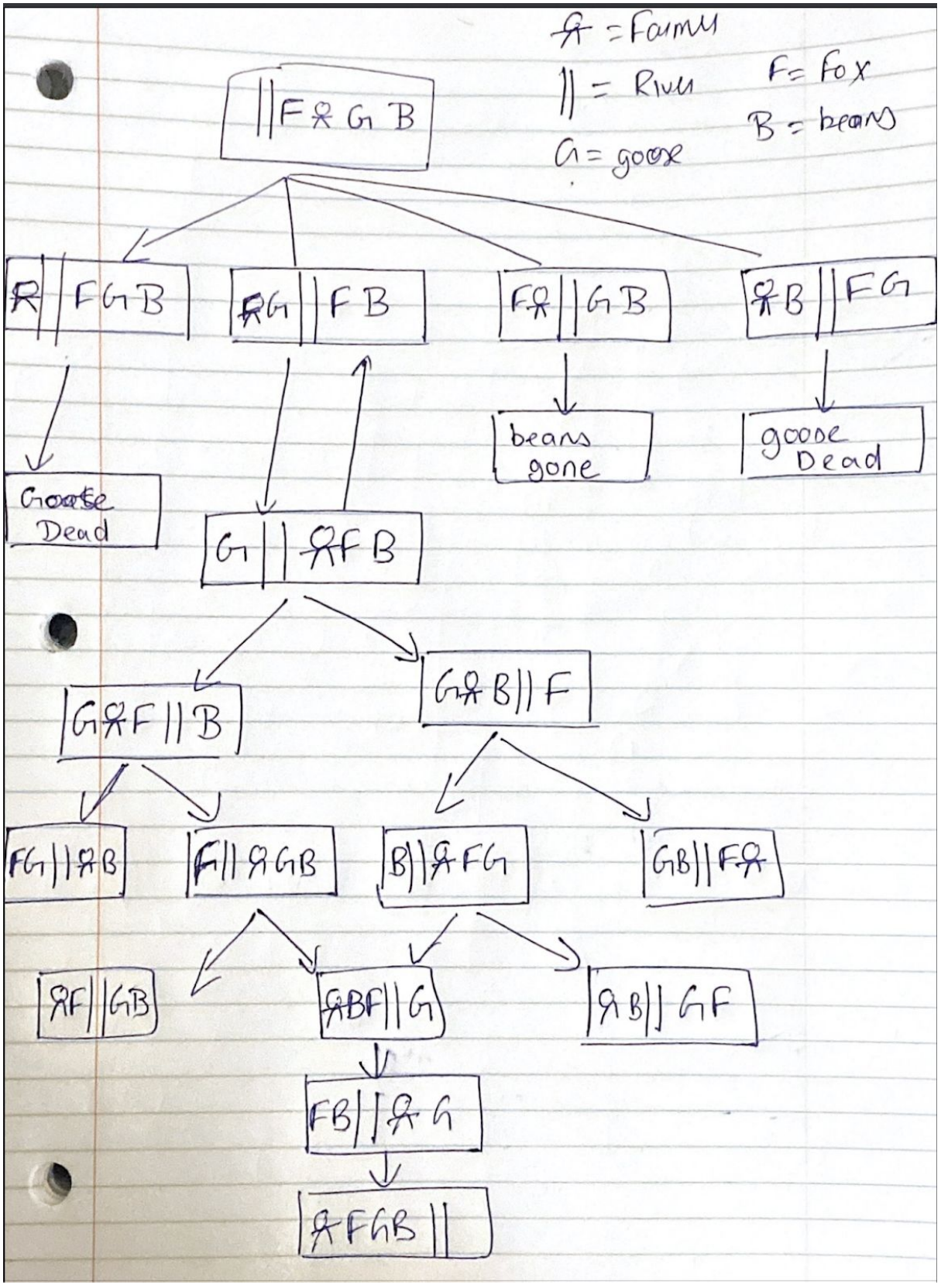
Once upon a time a farmer went to market and purchased a fox, a goose, and a bag of beans. On his way home, the farmer came to the bank of a river and hired a boat. But in crossing the river by boat, the farmer could carry only himself and a single one of his purchases - the fox, the goose, or the bag of the beans.

If left alone, the fox would eat the goose, and the goose would eat the beans.

The farmer's challenge was to carry himself and his purchases to the far bank of the river, leaving each purchase intact. How did he do it?

Use state-space-search method to solve this problem. Draw EXPLICITLY the state space and give the solution based on both DFS and BFS.

Solution ->



DFS : States that will be visited

1. {River - Farmer - Goose - Fox - Beans}

2. {Farmer - Goose - River -Fox - Beans}
3. {Goose - River - Farmer - Fox - Beans }
4. {Goose - Farmer - Fox - River - Beans}
5. {Fox - River Farmer - Goose - Beans}
6. {Farmer - Beans - Fox - River - Goose}
7. {Fox - Beans - River - Farmer - Goose}
8. {Fox - Beans - Farmer - Goose - River}

BFS : States that will be visited

1. {River - Farmer - Goose - Fox - Beans}
2. {Farmer - Goose - River -Fox - Beans}
3. {Goose - River - Farmer - Fox - Beans }
4. {Goose - Farmer - Fox - River - Beans}
5. {Goose - Farmer - Beans - River - Fox}
6. {Fox - River - Farmer - Goose - Beans}
7. {Beans - River - Farmer - Goose - Beans}
8. {Farmer - Beans - Fox - River - Goose}
9. {Fox - Beans - River - Farmer - Goose}
10. {Fox - Beans - Farmer - Goose - River}

3. (10 marks)
Consider Figure 3.29 (Page 122 of Textbook).

3.1. Suppose N is a goal state.
Give the list of nodes visited and a solution by using
DFS and BFS, respectively.

Solution -
BFS :
Solution 1 -
Nodes Visisted - A, B, C, D, E, F, G, H, I, J, K, L , M, N
Solution - A -> B -> G -> N

Solution 2 -
Nodes Visited - A, B, C, D, E, F, G, H, I, J, K, L , M, N, C, G, N
Solution - A -> C -> G -> N

DFS -
Nodes Visited - A,B, E, J, K, L, F, G, H, O, P, M, N
Solution- A -> B - > G -> N

3.2. Suppose P is a goal state.
Give the list of nodes visited and a solution by using
DFS and BFS, respectively.

Solution -
BFS :

Nodes Visited - A, B, C, D, E, F, G, H, I , J, K, L, M, N , O, P
Solution - A -> D -> H -> P

DFS -
Nodes Visited - A,B, E, J, K, L, F, G, H, O, P
Solution- A -> B - > G -> H -> P

4. (40 marks)

4.1. Implement the DFS algorithm by using any language of your choice.
Solution -

```
graph = {
   'A' : ['B','C','D'],
   'B' : ['E','F','G'],
   'C' : ['G'],
   'D' : ['H','I'],
   'E' : ['J','K','L'],
   'F' : ['A','L'],
```

```
    'G' : ['H','M','N'],
    'H' : ['O','P','A'],
    'I' : ['P','R'],
    'J' : [],
    'K' : [],
    'L' : [],
    'M' : [],
    'N' : [],
    'O' : [],
    'P' : [],
    'R' : [],

}

def dfs(graph, node, visited):
    if node not in visited:
        visited.append(node)
        print(visited)
        for n in graph[node]:
            dfs(graph,n, visited)
    return visited

visited = dfs(graph,'A', [])
```

```python
graph = {
    'A' : ['B','C','D'],
    'B' : ['E','F','G'],
    'C' : ['G'],
    'D' : ['H','I'],
    'E' : ['J','K','L'],
    'F' : ['A','L'],
    'G' : ['H','M','N'],
    'H' : ['O','P','A'],
    'I' : ['P','R'],
    'J' : [],
    'K' : [],
    'L' : [],
    'M' : [],
    'N' : [],
    'O' : [],
    'P' : [],
    'R' : [],

}

def dfs(graph, node, visited):
    if node not in visited:
        visited.append(node)
        print(visited)
        for n in graph[node]:
            dfs(graph,n, visited)
    return visited

visited = dfs(graph,'A', [])
```

4.2. Implement the BFS algorithm by using any language of your choice.

```
path_queue = MyQUEUE() # now we make a queue


def BFS(graph,start,end,q):

    temp_path = [start]

    q.enqueue(temp_path)

    while q.IsEmpty() == False:
        tmp_path = q.dequeue()
        last_node = tmp_path[len(tmp_path)-1]
        print (tmp_path)
        if last_node == end:
            print ("VALID_PATH : ",tmp_path)
        for link_node in graph[last_node]:
            if link_node not in tmp_path:
                new_path = []
                new_path = tmp_path + [link_node]
                q.enqueue(new_path)

BFS(graph,"A","P",path_queue)
```

```
# a sample graph
graph = {
   'A' : ['B','C','D'],
   'B' : ['E','F','G'],
   'C' : ['G'],
   'D' : ['H','I'],
   'E' : ['J','K','L'],
   'F' : ['A','L'],
   'G' : ['H','M','N'],
   'H' : ['O','P','A'],
```

```
    'I' : ['P','R'],
    'J' : [],
    'K' : [],
    'L' : [],
    'M' : [],
    'N' : [],
    'O' : [],
    'P' : [],
    'R' : [],

}

class MyQUEUE: # just an implementation of a queue

    def __init__(self):
        self.holder = []

    def enqueue(self,val):
        self.holder.append(val)

    def dequeue(self):
        val = None
        try:
            val = self.holder[0]
            if len(self.holder) == 1:
                self.holder = []
            else:
                self.holder = self.holder[1:]
        except:
            pass

        return val

    def IsEmpty(self):
        result = False
```

```
        if len(self.holder) == 0:
            result = True
        return result


path_queue = MyQUEUE() # now we make a queue


def BFS(graph,start,end,q):

    temp_path = [start]

    q.enqueue(temp_path)

    while q.IsEmpty() == False:
        tmp_path = q.dequeue()
        last_node = tmp_path[len(tmp_path)-1]
        print (tmp_path)
        if last_node == end:
            print ("VALID_PATH : ",tmp_path)
        for link_node in graph[last_node]:
            if link_node not in tmp_path:
                new_path = []
                new_path = tmp_path + [link_node]
                q.enqueue(new_path)

BFS(graph,"A","P",path_queue)
```

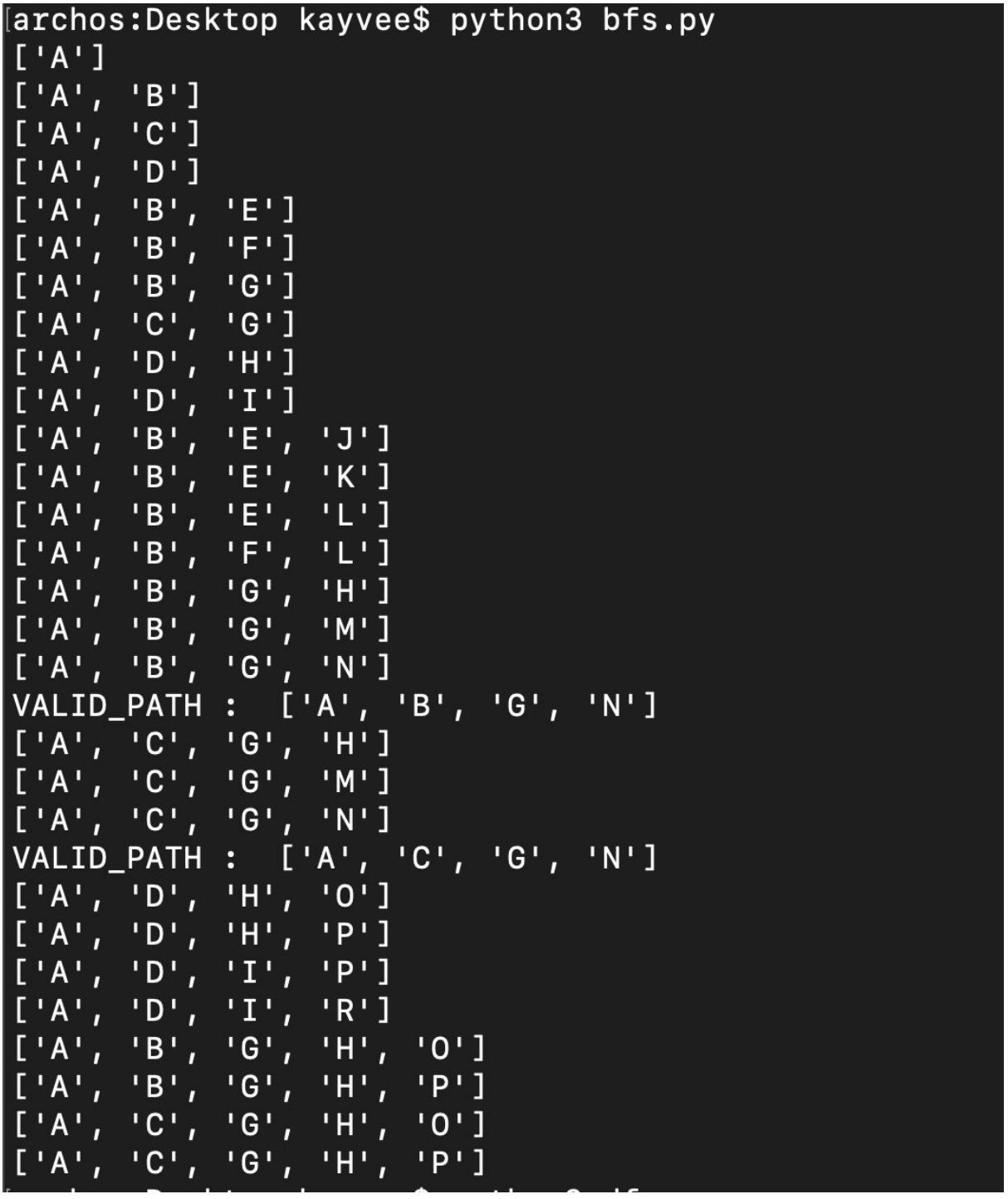
4.3. Test your program by using Question 3.

Solution -

Screenshot for finding P

```
archos:Desktop kayvee$ python3 bfs.py
['A']
['A', 'B']
['A', 'C']
['A', 'D']
['A', 'B', 'E']
['A', 'B', 'F']
['A', 'B', 'G']
['A', 'C', 'G']
['A', 'D', 'H']
['A', 'D', 'I']
['A', 'B', 'E', 'J']
['A', 'B', 'E', 'K']
['A', 'B', 'E', 'L']
['A', 'B', 'F', 'L']
['A', 'B', 'G', 'H']
['A', 'B', 'G', 'M']
['A', 'B', 'G', 'N']
['A', 'C', 'G', 'H']
['A', 'C', 'G', 'M']
['A', 'C', 'G', 'N']
['A', 'D', 'H', 'O']
['A', 'D', 'H', 'P']
VALID_PATH :  ['A', 'D', 'H', 'P']
['A', 'D', 'I', 'P']
VALID_PATH :  ['A', 'D', 'I', 'P']
['A', 'D', 'I', 'R']
['A', 'B', 'G', 'H', 'O']
['A', 'B', 'G', 'H', 'P']
VALID_PATH :  ['A', 'B', 'G', 'H', 'P']
['A', 'C', 'G', 'H', 'O']
['A', 'C', 'G', 'H', 'P']
VALID_PATH :  ['A', 'C', 'G', 'H', 'P']
```

Screenshot for finding P



DFS implementation

```
archos:Desktop kayvee$ python3 dfs.py
['A']
['A', 'B']
['A', 'B', 'E']
['A', 'B', 'E', 'J']
['A', 'B', 'E', 'J', 'K']
['A', 'B', 'E', 'J', 'K', 'L']
['A', 'B', 'E', 'J', 'K', 'L', 'F']
['A', 'B', 'E', 'J', 'K', 'L', 'F', 'G']
['A', 'B', 'E', 'J', 'K', 'L', 'F', 'G', 'H']
['A', 'B', 'E', 'J', 'K', 'L', 'F', 'G', 'H', 'O']
['A', 'B', 'E', 'J', 'K', 'L', 'F', 'G', 'H', 'O', 'P']
['A', 'B', 'E', 'J', 'K', 'L', 'F', 'G', 'H', 'O', 'P', 'M']
['A', 'B', 'E', 'J', 'K', 'L', 'F', 'G', 'H', 'O', 'P', 'M', 'N']
['A', 'B', 'E', 'J', 'K', 'L', 'F', 'G', 'H', 'O', 'P', 'M', 'N', 'C']
['A', 'B', 'E', 'J', 'K', 'L', 'F', 'G', 'H', 'O', 'P', 'M', 'N', 'C', 'D']
['A', 'B', 'E', 'J', 'K', 'L', 'F', 'G', 'H', 'O', 'P', 'M', 'N', 'C', 'D', 'I']
['A', 'B', 'E', 'J', 'K', 'L', 'F', 'G', 'H', 'O', 'P', 'M', 'N', 'C', 'D', 'I', 'R']
archos:Desktop kayvee$
```