Group : ENIGMA
Due: Wed 23 Oct 2019

CS 461 – ARTIFICIAL INTELLIGENCE

HOMEWORK #1

Group ENIGMA

Members:

Özlem Tutku Naz Demirbaş	21501611
Mehmet Mert Epsileli	21502933
Güner Eda Turanlı	21503140
Batuhan Ünal	21502755
Eren Yalçın	21602004

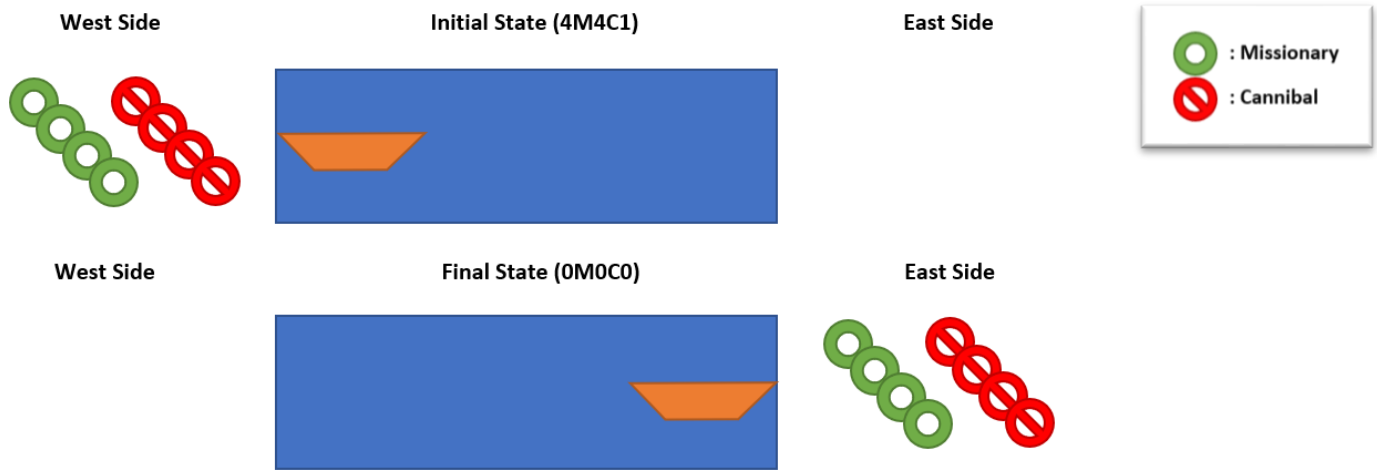

Figure 1: Demonstration of the initial and goal (final) states

Our aim in this homework is to find all paths for the problem demonstrated in Figure 1 even though the goal state is not reached. Initial state consists of 4 cannibals and 4 missionaries on the west side of a river and the goal state is to transport 4 cannibals and 4 missionaries to the east side by a boat that can carry up to 2 persons at a time. The safe conditions appear to be the conditions that missionaries are not outnumbered by the cannibals on both sides of the river.

We used the state representation in the form xMyCb where x represented the number of missionaries, y represented the number of cannibals and b is the side of boat (0 if it is on the east, 1 if it is on the west side). The initial state is then: 4M4C1 and the goal state is: 0M0C0.

The possible children of a node with representation (x, y, b) are (x-2, y, b=0), (x-1, y, b=0), (x-1, y-1, b=0), (x, y-1, b=0) and (x, y-2, b=0) if the boat currently represented to be in the west and (x+2, y, b=1), (x+1, y, b=1), (x+1, y+1, b=1), (x, y+1, b=1) and (x, y+2, b=1) if the boat is in the east side of

the river. According to these children the operators are 2M, 1M, 1M1C, 1C and 2C. They represent the number of missionaries and cannibals boarded on the boat to travel to the other side of the river. So, for each consecutive state the side of boat changes and b flips. Then we can easily observe that there is no constant number of children that is valid for all nodes since there is unsafe and impossible states such that 0M0C1 and the loops are prevented. Therefore, the branching factor changes between 1-5 theoretically and the trees branching factor is asymptotic. Furthermore , we can also say that no node has 5 children since even in the first state there is no cannibal nor missionary in the east side of the river so that if one cannibal or missionary goes alone to the east side the same one needs to return the boat back. There happens to be a loop, so we ignore these two children.

Our approach was to deliver all possible paths even though they do not reach the goal state. We have written two different codes by using Depth First Search and Breadth First Search separately.

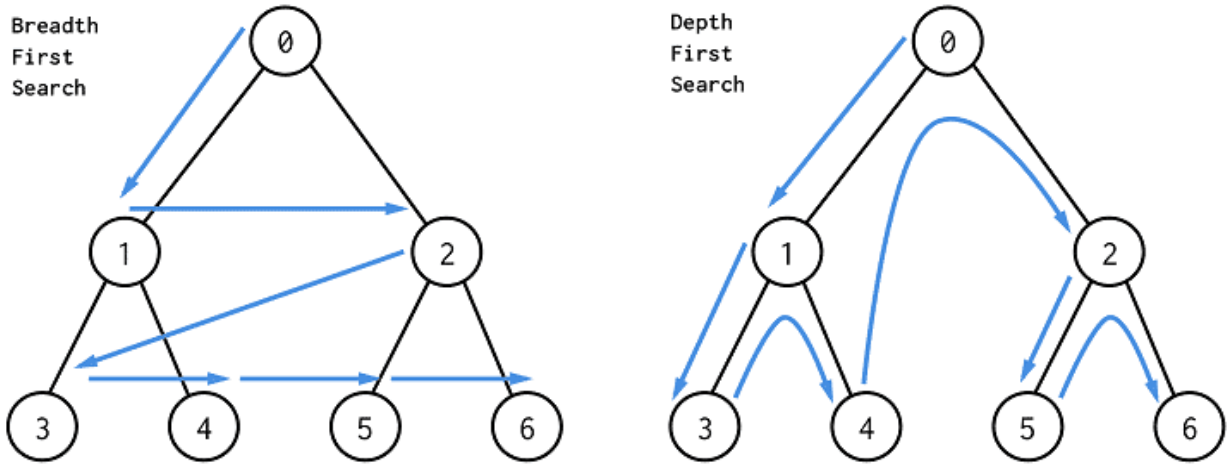

Figure 2: Demonstration of the path in BFS and DFS Algorithms. **[1]**

Breadth First Search (BFS)

BFS is an algorithm that is used for traversing or searching a tree or graph data structures **[2]**. BFS starts at the first node, which can also be called the root node, and goes through the neighbor or sibling nodes first. When the algorithm reaches the last node at the present depth, it moves to the nodes on the next depth level and continues until it reaches the last on the last depth level. For the tree that is demonstrated in Figure 2, the stack will be created as 0-1-2-3-4-5-6.

Depth First Search (DFS)

DFS algorithm is an algorithm to search or traverse a tree or graph data structures **[3]**. The algorithm starts from the first node or the root node and goes in one path until it reaches the last node in that path. When the last node is reached and the path can no longer be followed, the algorithm starts backtracking until a new path is available and follows them until the last node. For the tree that is demonstrated in Figure 2, the stack will be created as 0-1-3-4-2-5-6.

Output of the Program

```
Graph representation
{441=[430, 420, 330], 330=[431, 441], 401=[], 011=[000], 220=[421, 331], 030=[041], 040=[], 110=[221], 410=[421, 431], 400=[411,
421], 010=[111, 021, 031], 411=[400], 421=[220, 410, 400], 001=[], 221=[020, 110], 431=[330, 420, 410], 000=[011, 021, 111], 111=
[010, 000], 440=[], 021=[010, 000], 331=[220], 020=[221, 031, 041], 420=[431, 441], 031=[020, 010], 430=[441], 041=[030, 020]}

Tree representation

Layer 0
4M4C1

Layer 1
4M3C0 4M2C0 3M3C0

Layer 2
4M4C1(repetition) 4M3C1 4M4C1(repetition) 4M3C1(repetition) 4M4C1(repetition)

Layer 3
3M3C0(repetition) 4M2C0(repetition) 4M1C0

Layer 4
4M2C1 4M3C1(repetition)

Layer 5
2M2C0 4M1C0(repetition) 4M0C0

Layer 6
4M2C1(repetition) 3M3C1 4M1C1 4M2C1(repetition)

Layer 7
2M2C0(repetition) 4M0C0(repetition)

Searching Algorithms' Results
Breadth First Search Result -> [4M4C1, 4M3C0, 4M2C0, 3M3C0, 4M3C1, 4M1C0, 4M2C1, 2M2C0, 4M0C0, 3M3C1, 4M1C1]
Depth First Search Result -> [4M4C1, 3M3C0, 4M3C1, 4M1C0, 4M2C1, 4M0C0, 4M1C1, 2M2C0, 3M3C1, 4M2C0, 4M3C0]


Paths
[4M4C1, 3M3C0, 4M4C1]->Repetition
[4M4C1, 3M3C0, 4M3C1, 4M1C0, 4M3C1]->Repetition
[4M4C1, 3M3C0, 4M3C1, 4M1C0, 4M2C1, 4M0C0, 4M2C1]->Repetition
[4M4C1, 3M3C0, 4M3C1, 4M1C0, 4M2C1, 4M0C0, 4M1C1, 4M0C0]->Repetition
[4M4C1, 3M3C0, 4M3C1, 4M1C0, 4M2C1, 4M1C0]->Repetition
[4M4C1, 3M3C0, 4M3C1, 4M1C0, 4M2C1, 2M2C0, 3M3C1, 2M2C0]->Repetition
[4M4C1, 3M3C0, 4M3C1, 4M1C0, 4M2C1, 2M2C0, 4M2C1]->Repetition
[4M4C1, 3M3C0, 4M3C1, 4M2C0, 4M4C1]->Repetition
[4M4C1, 3M3C0, 4M3C1, 4M2C0, 4M3C1]->Repetition
[4M4C1, 3M3C0, 4M3C1, 3M3C0]->Repetition
[4M4C1, 3M3C0, 4M3C1, 4M2C0]->Repetition
[4M4C1, 4M3C0, 4M4C1]->Repetition
```

APPENDIX

State.java

```
public class State {
		boolean safe;
		int x; //Number of Missionaries
		int y; //Number of Cannibals
		int b; //Side of boat 1=west 0=east

		public State(int missionary, int cannibal, int boatSide)
		{
			if( missionary >= 0 && cannibal >= 0) {
				x = missionary;
				y = cannibal;
				b = boatSide;

				if (missionary == 0 || missionary == 4)
					safe = true;
				else if(missionary == cannibal)
					safe = true;
				else
					safe = false;
			}
		}

		@Override
		public String toString()
		{
```

```
		return "" + x + y + b;
	}
}
```

States.java

```
public class States {
	State allStates[] = new State[50];

	public States() {
		int place = 0;
		for(int missionary = 0; missionary <= 4; missionary++)
		{
			for(int cannibal = 0; cannibal <= 4; cannibal++)
			{
				for(int boatSide = 0; boatSide <= 1; boatSide++)
				{
					allStates[place] = new State(missionary, cannibal, boatSide);
					place++;
				}
			}
		}
	}

	State getState(int missionary, int cannibal, int boatSide)
	{
		for(State aState : allStates)
		{
```

```
                    if(aState.x == missionary && aState.y == cannibal && aState.b == boatSide)
                        return aState;
                }
                return null;
            }

}
```

Graph.java

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map.Entry;

public class Graph {
        HashMap<State, ArrayList<State>> stateGraph;

        public Graph() {
            stateGraph = new HashMap<State, ArrayList<State>>();
        }

        void addState(State newState)
        {
            stateGraph.putIfAbsent(newState, new ArrayList<>());
        }

        void addEdge(State state1, State state2)
        {
```

```
            stateGraph.get(state1).add(state2);
        }

        ArrayList<State> getEdgedStates(State aState)
        {
            return stateGraph.get(aState);
        }
}
```

Homework1.java

```
import java.util.*;



public class Homework1 {

        public static void main(String[] args) {
            Graph graph = new Graph();
            States states = new States();

            //Add States to Graph
            for(int missionary = 0; missionary <= 4; missionary++)
            {
                for(int cannibal = 0; cannibal <= 4; cannibal++)
                {
                    for(int boatSide = 0; boatSide <= 1; boatSide++)
                    {
                        if(states.getState(missionary, cannibal, boatSide).safe )
```

```
                                        graph.addState(states.getState(missionary,
cannibal, boatSide));
                                }
                        }
                }

                //Add Edges between States
                for(int missionary = 0; missionary <= 4; missionary++)
                {
                        for(int cannibal = 0; cannibal <= 4; cannibal++)
                        {
                                for(int boatSide = 0; boatSide <= 1; boatSide++)
                                {
                                        State                state1                =
states.getState(missionary,cannibal,boatSide);;
                                        if(boatSide == 0)
                                        {
                                                if( missionary <= 3) {
                                                        State                state2                =
states.getState(missionary + 1,cannibal,boatSide + 1);
                                                        if(state1.safe && state2.safe)
                                                                graph.addEdge(state1, state2);
                                                }

                                                if( missionary <= 2) {
                                                        State                state2                =
states.getState(missionary + 2,cannibal,boatSide + 1);
                                                        if(state1.safe && state2.safe)
                                                                graph.addEdge(state1, state2);
                                                }
```

```
                    if( cannibal <= 3) {
                        State state2 = states.getState(missionary,cannibal + 1,boatSide + 1);
                        if(state1.safe && state2.safe)
                            graph.addEdge(state1, state2);
                    }

                    if( cannibal <= 2) {
                        State state2 = states.getState(missionary,cannibal + 2,boatSide + 1);
                        if(state1.safe && state2.safe)
                            graph.addEdge(state1, state2);
                    }

                    if( cannibal <= 3 && missionary <= 3) {
                        State state2 = states.getState(missionary + 1,cannibal + 1,boatSide + 1);
                        if(state1.safe && state2.safe)
                            graph.addEdge(state1, state2);
                    }
                }

                if(boatSide == 1)
                {
                    if( missionary >= 1) {
                        State state2 = states.getState(missionary - 1,cannibal,boatSide - 1);
                        if(state1.safe && state2.safe)
                            graph.addEdge(state1, state2);
```

```
                }

                if( missionary >= 2) {
                        State state2 =
states.getState(missionary - 2,cannibal,boatSide - 1);
                        if(state1.safe && state2.safe)
                                graph.addEdge(state1, state2);
                }

                if( cannibal >= 1) {
                        State state2 =
states.getState(missionary,cannibal - 1,boatSide - 1);
                        if(state1.safe && state2.safe)
                                graph.addEdge(state1, state2);
                }

                if( cannibal >= 2) {
                        State state2 =
states.getState(missionary,cannibal - 2,boatSide - 1);
                        if(state1.safe && state2.safe)
                                graph.addEdge(state1, state2);
                }

                if( cannibal >= 1 && missionary >= 1) {
                        State state2 =
states.getState(missionary - 1,cannibal - 1,boatSide - 1);
                        if(state1.safe && state2.safe)
                                graph.addEdge(state1, state2);
                }
        }
```

```
                    }
                }
            }

            System.out.println("Graph representation \n" + graph.stateGraph.toString());
            System.out.println("\nTree representation");
            System.out.println("Searching Algorithms' Results \nBreadth First Search Result -> " + breadthFirstSearch(graph,states,states.getState(4, 4, 1)).toString());
            System.out.println("Depth First Search Result -> " + depthFirstSearch(graph,states,states.getState(4, 4, 1)));
            System.out.println("\nPaths");
            printPaths(graph,states,states.getState(4, 4, 1));
        }

        static ArrayList<String> depthFirstSearch(Graph graph,States states, State initial)
        {
            ArrayList<String> visited = new ArrayList<String>();
            Stack<String> stateRepresentation = new Stack<String>();
            String representation = initial.x + "M" + initial.y + "C" + initial.b;
            stateRepresentation.push(representation);

            while(!stateRepresentation.empty())
            {
                String stateString = stateRepresentation.pop();
                if(!visited.contains(stateString)) {
                    visited.add(stateString);
                    for(State aState : graph.getEdgedStates(states.getState(Character.getNumericValue(stateString.charAt(0)), Character.getNumericValue(stateString.charAt(2)), Character.getNumericValue(stateString.charAt(4)))))
                    {
```

```
                                        stateRepresentation.push(aState.x + "M" + aState.y + "C"
+ aState.b);
                                    }
                                }
                            }
                            return visited;
                        }

                        static void printPaths(Graph graph,States states, State initial)
                        {
                            ArrayList<String> visited = new ArrayList<String>();
                            Stack<String> stateRepresentation = new Stack<String>();
                            String representation = initial.x + "M" + initial.y + "C" + initial.b;
                            stateRepresentation.push(representation);

                            while(!stateRepresentation.empty())
                            {
                                String stateString = stateRepresentation.pop();
                                if(!visited.contains(stateString)) {
                                    if(!visited.isEmpty()) {
                                        String currentLastState = visited.get(visited.size()-1);
                                        ArrayList<State> edgedStates =
graph.getEdgedStates(states.getState(Character.getNumericValue(currentLastState.charAt(0)),
Character.getNumericValue(currentLastState.charAt(2)),
Character.getNumericValue(currentLastState.charAt(4))));

                        while(!edgedStates.contains(states.getState(Character.getNumericValue(stateString.c
harAt(0)), Character.getNumericValue(stateString.charAt(2)),
Character.getNumericValue(stateString.charAt(4)))))
                                        {
                                            visited.remove(visited.size()-1);
```

```
                                        currentLastState = visited.get(visited.size()-1);

                                        edgedStates =
graph.getEdgedStates(states.getState(Character.getNumericValue(currentLastState.charAt(0)),
Character.getNumericValue(currentLastState.charAt(2)),
Character.getNumericValue(currentLastState.charAt(4))));
                                    }

                                }
                                visited.add(stateString);
                                for(State aState :
graph.getEdgedStates(states.getState(Character.getNumericValue(stateString.charAt(0)),
Character.getNumericValue(stateString.charAt(2)), Character.getNumericValue(stateString.charAt(4)))))
                                {
                                    stateRepresentation.push(aState.x + "M" + aState.y + "C"
+ aState.b);
                                }
                            }
                            else
                            {
                                ArrayList<String> visitedTemp = new ArrayList<String>();
                                visitedTemp.addAll(visited);
                                String currentLastState = visitedTemp.get(visitedTemp.size()-1);
                                ArrayList<State> edgedStates =
graph.getEdgedStates(states.getState(Character.getNumericValue(currentLastState.charAt(0)),
Character.getNumericValue(currentLastState.charAt(2)),
Character.getNumericValue(currentLastState.charAt(4))));

                                while(!edgedStates.contains(states.getState(Character.getNumericValue(stateString.c
harAt(0)), Character.getNumericValue(stateString.charAt(2)),
Character.getNumericValue(stateString.charAt(4)))))
                                {
                                    visitedTemp.remove(visitedTemp.size()-1);
```

```
                                        currentLastState = visitedTemp.get(visitedTemp.size()-
1);
                                        edgedStates =
graph.getEdgedStates(states.getState(Character.getNumericValue(currentLastState.charAt(0)),
Character.getNumericValue(currentLastState.charAt(2)),
Character.getNumericValue(currentLastState.charAt(4))));
                                }
                                visitedTemp.add(stateString);
                                System.out.println(visitedTemp.toString() + "->Repetition");
                        }

                }
        }

        static Set<String> breadthFirstSearch(Graph graph,States states, State initial)
        {
                int layerCnt = 0;
                int currState = initial.b;

                Set<String> visited = new LinkedHashSet<String>();
                Queue<String> stateRepresentation = new LinkedList<String>();
                String representation = initial.x + "M" + initial.y + "C" + initial.b;
                stateRepresentation.add(representation);
                visited.add(representation);
                System.out.println("\nLayer " + layerCnt);
                System.out.print(representation + " ");

                while(!stateRepresentation.isEmpty())
                {
                        String stateString = stateRepresentation.poll();
```

```
for(State aState : graph.getEdgedStates(states.getState(Character.getNumericValue(stateString.charAt(0)), Character.getNumericValue(stateString.charAt(2)), Character.getNumericValue(stateString.charAt(4)))))
{
    String newState = aState.x + "M" + aState.y + "C" + aState.b;
    if(!visited.contains(newState)) {
        if(aState.b == currState) {
            visited.add(newState);
            System.out.print(newState + " ");
            stateRepresentation.add(newState);
        }
        else {
            currState = aState.b;
            layerCnt++;
            System.out.println("\n\nLayer " + layerCnt);
            visited.add(newState);
            System.out.print(newState + " ");
            stateRepresentation.add(newState);
        }
    } else {
        if(aState.b == currState) {
            System.out.print(newState +  "(repetition) ");
        }
        else {
            currState = aState.b;
            layerCnt++;
            System.out.println("\n\nLayer " + layerCnt);
            System.out.print(newState + "(repetition) ");
        }
```

```
                    }
                }
            }
            System.out.print("\n\n");
            return visited;
        }
}
```

REFERENCES

[1] K. Gupta, "What is the Difference Between BFS and DFS Algorithms," *Web, Design, Programming*, 06-Feb-2019. [Online]. Available: https://www.freelancinggig.com/blog/2019/02/06/what-is-the-difference-between-bfs-and-dfs-algorithms/. [Accessed: 23-Oct-2019].

[2] C. to W. projects, "algorithm for searching the nodes of a graph in order by their hop count from a starting node," *Wikipedia*, 22-Oct-2019. [Online]. Available: https://www.wikizeroo.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvQnJlYWR0aC1maXJzdF9zZWFyY2g. [Accessed: 23-Oct-2019].

[3] C. to W. projects, "search algorithm," *Wikipedia*, 15-Oct-2019. [Online]. Available: https://www.wikizeroo.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvRGVwdGgtZmlyc3Rfc2VhcmNo. [Accessed: 23-Oct-2019].