Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Assignment Report

Submitted by:

Saskar Khadka (27)

Chandan Kumar Mahato (31)

Submitted to:

Mr. Santosh Khanal

Department of Computer Science and Engineering

Submission Date: 5th Jan, 2024

ACKNOWLEDGEMENT

We would like to express our gratitude for your guidance and support throughout the completion of the assignment on State Space Search, specifically focusing on the Missionaries and Cannibals Problem, as well as exploration of the 8 Puzzle Game using missing tiles heuristics.

The assignment allowed us to delve deeply into fundamental problem-solving techniques ad heuristic strategies, broadening our understanding of state representation and exploring various search algorithm like Breadth-first Search (BFS), Depth-first Search (DFS) and A* in depth.

We are grateful for the opportunity to engage with such intellectually stimulating content and are appreciative of your valuable feedback and insight that have significantly contributed to our learning experience.

Contents

Introduction **4**

Q1. Missionaries and Cannibal Game **4**

1.1 Introduction 4

1.2 Game Components 4

1.2.1 State Representation 4

1.2.2 Valid Actions 4

1.2.3 Rules 5

1.3 Source Code 5

1.4 Output 17

1.5 Animation: 18

Q2. Explore the Solution of Q.No.1 using Breadth-first Search **18**

2.1 Introduction 18

2.2 Source Code 19

2.3 Screenshots 24

Q3. 8 Puzzle Game Using A* with heuristics (misplaced tiles) **26**

3.1 Introduction 26

3.2 Game Components: 26

3.2.1 Configuration 26

3.2.2 Rules 26

3.3 Using A-Star 26

3.3.1 Problem Definition & State Representation 26

3.4 Source Code 27

3.5 Output 36

Introduction

Artificial Intelligence (AI) refers to the development of computer systems that can perform tasks that typically require human intelligence. These tasks include learning, reasoning, problem-solving, perception, language understanding, and even speech recognition. AI aims to create machine that can simulate human cognitive functions and adapt to new situations.

Q1. Missionaries and Cannibal Game

State Space Search

Represent the state space for the well-known Missionaries and Cannibals Game as [3,3,0] where the first number represent the number of cannibals, second number represent the number of cannibals and the third number represent the left bank or right bank. Define the problem and generate the tree like structure which represent the state space.

1.1 Introduction

In the missionaries and cannibals' problem, three missionaries and three cannibals must cross a river using a boat which can carry a most two people, under the constraint that, for both banks, if there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries). The boat cannot cross the river by itself with no people on board.

1.2 Game Components

1.2.1 State Representation

A state can be represented by the number of missionaries and cannibals on each side of the river.

- Initial state: **3m, 3c, canoe**
- Goal state: **0m, 0c**

1.2.2 Valid Actions

Now we can choose to occupy the boat with following option

- 1 cannibal
- 1 missionaries
- 2 cannibals
- 2 missionaries
- 1 cannibal and 1 missionaries

1.2.3 Rules

Now while traveling through the boat following things must be considered

- Number of missionaries must always be greater or equal to cannibal in either side of the river
- Boat can't be moved when empty

An operation takes us from one state to another, and this process is shown below using a tree like structure which has been generated programmatically from the implementation of Missionaries and Cannibals Problem using JavaScript, and graphing library Vis.js.

1.3 Source Code

Breakdown of start.js

Following code performs a breadth-first search (BFS) algorithm to find a solution to the missionary and cannibal problem, a classic puzzle in artificial intelligence.

- **Node Class:** Represents a state in the problem space. It defines functions to check if it's a goal state, if it's valid, generates child states, and finds a solution path. It explores possible moves from each state, keeping track of explored, opened, and killed nodes. It stops if it finds a solution or exhausts all possibilities.

- **BFS Algorithm:** The 'bfs' function runs the breadth-first search algorithm. It starts with an initial state and explores possible states systematically, considering valid moves and avoiding scenarios where missionaries are outnumbered by cannibals.

- **Solution Extraction and Data Storage:** Once a solution is found it records the solution path, killed nodes, and the traversal tree into respective data structures. It then saves these data structures into text files and triggers the execution of scripts **(`drawGraph.js`** and UI-related scripts) to visualize the solution and graphically represent the problem.

start.js

```
const fs = require("fs");
const { exec } = require("child_process");
const { createScriptFile } = require("./UI");

let tree = [];
let from = [];
let to = [];
let weight = [];

const freezeTime = 100;
```

```
function delay(time) {
  return new Promise((resolve) => setTimeout(resolve, time));
}
class Node {
  constructor(state, parent, action, depth) {
    this.parent = parent;
    this.state = state;
    this.action = action;
    this.depth = depth;
  }
  isGoal() {
    if (this.state[0] == 0 && this.state[1] == 0 && this.state[2] == 0) {
      return true;
    }
    return false;
  }
  isValid() {
    let missionaries = this.state[0];
    let cannibals = this.state[1];
    let boat = this.state[2];

    if (missionaries < 0 || missionaries > 3) {
      return false;
    }

    if (cannibals < 0 || cannibals > 3) {
      return false;
    }

    if (boat > 1 || boat < 0) {
      return false;
    }
    return true;
  }
  mustBeKilled() {
    let missionaries = this.state[0];
    let cannibals = this.state[1];

    if (missionaries < cannibals && missionaries > 0) {
      return true;
    }

    if (missionaries > cannibals && missionaries < 3) {
      return true;
    }
```

```
  }

  generateChild() {
    let children = [];
    let canoe = -1;
    if (this.state[2] == 0) {
      canoe = 1;
    }
    for (let x = 0; x < 3; x++) {
      for (let y = 0; y < 3; y++) {
        let newState = this.state.slice();
        newState[0] = newState[0] + canoe * x;
        newState[1] = newState[1] + canoe * y;
        newState[2] = newState[2] + canoe * 1;
        let action = [x, y, canoe];
        const newNode = new Node(newState, this, action, this.depth + 1);
        if (x + y >= 1 && x + y <= 2) {
          children.push(newNode);
        }
      }
    }
    return children;
  }
  findSolution() {
    let solution = [];
    solution.push(this.action);
    let path = new Node(this.state, this.parent, this.action);
    while (path.parent != null) {
      path = path.parent;
      solution.push(path.action);
    }
    solution.pop();
    solution.reverse();
    return solution;
  }
}
function existsIn(main, check) {
  for (i = 0; i < main.length; i++) {
    if (
      main[i][0] == check[0] &&
      main[i][1] == check[1] &&
      main[i][2] == check[2]
    ) {
      return true;
    }
```

```
  }
  return false;
}
async function bfs(initialState) {
  let startNode = new Node(initialState, null, null, 0);

  if (startNode.isGoal()) {
    return startNode.findSolution();
  }
  let q = [];
  let q_states = [];
  q.push(startNode);
  q_states.push(startNode.state);
  let explored = [];
  let killed = [];
  let opened = [];
  console.log(`The initial state is ${startNode.state}`);
  while (!(q.length == 0)) {
    console.log(`\nBFS Queue: ${JSON.stringify(q_states)}`);
    await delay(freezeTime).then(() => {});
    let node = q.shift();
    q_states.shift();
    console.log(`\nThe node to expand is ${node.state}\n`);
    await delay(freezeTime).then(() => {});
    explored.push(node.state);
    let children = node.generateChild();
    if (!node.mustBeKilled()) {
      opened.push(node.state);
      console.log(`The valid children of ${node.state}:`);
      await delay(freezeTime).then(() => {});
      let count = 0;
      for (const child of children) {
        if (!existsIn(explored, child.state)) {
          if (child.isValid()) {
            count++;
            q.push(child);
            q_states.push(child.state);
            from.push([node.state, node.depth]);
            to.push([child.state, child.depth]);
            weight.push(child.action);
            explored.push(child.state);
            console.log(child.state);
            if (child.isGoal()) {
              console.log("Goal State Found");
              await delay(freezeTime).then(() => {});
```

```
              console.log("\n");
              return {
                solution: child.findSolution(),
                killedNodes: killed,
                opened: opened,
              };
            }
          }

          await delay(freezeTime).then(() => {});
        }
      }
      if (count === 0) {
        console.log("The node cannot be further expanded");
        await delay(freezeTime).then(() => {});
      }
    } else {
      killed.push(node.state);
      console.log(
        "The state as respresented by this node is not possible. So it is
killed."
      );
      await delay(freezeTime).then(() => {});
    }
  }
}
async function bfsSolutionAndDataSaver() {
  let initialState = [3, 3, 1];
  const res = await bfs(initialState);
  console.log("Solution:");
  console.log(res["solution"]);
  for (let i = 0; i < to.length; i++) {
    let contains = false;
    for (const node of res["killedNodes"]) {
      if (
        to[i][0][0] == node[0] &&
        to[i][0][1] == node[1] &&
        to[i][0][2] == node[2]
      ) {
        contains = true;
        break;
      }
    }
    if (contains) {
      let appendData =
```

```
        JSON.stringify(from[i][0]) +
        " " +
        JSON.stringify(to[i][0]) +
        " " +
        JSON.stringify(weight[i]) +
        " " +
        "1" +
        " " +
        JSON.stringify(from[i][1]) +
        " " +
        JSON.stringify(to[i][1]) +
        "\n";
      tree.push(appendData);
    } else {
      if (
        existsIn(res["opened"], to[i][0]) ||
        (to[i][0][0] == "0" && to[i][0][1] == "0" && to[i][0][2] == "0")
      ) {
        let appendData =
          JSON.stringify(from[i][0]) +
          " " +
          JSON.stringify(to[i][0]) +
          " " +
          JSON.stringify(weight[i]) +
          " " +
          "0" +
          " " +
          JSON.stringify(from[i][1]) +
          " " +
          JSON.stringify(to[i][1]) +
          "\n";
        tree.push(appendData);
      } else {
        let appendData =
          JSON.stringify(from[i][0]) +
          " " +
          JSON.stringify(to[i][0]) +
          " " +
          JSON.stringify(weight[i]) +
          " " +
          "2" +
          " " +
          JSON.stringify(from[i][1]) +
          " " +
          JSON.stringify(to[i][1]) +
```

```
          "\n";
        tree.push(appendData);
      }
    }
}
function forFileWrite(arr) {
  let convertedData = [];
  for (data of arr) {
    let converted = JSON.stringify(data) + ",";
    convertedData.push(converted);
  }
  return convertedData;
}
function forSolutionWrite(arr) {
  let convertedData = [];
  for (data of arr) {
    let converted = JSON.stringify(data) + "\n";
    convertedData.push(converted);
  }
  return convertedData;
}
fs.writeFile("./StateSpace/data.txt", tree.join("").trim(), (err) => {
  if (err) {
    console.error(err);
  } else {
    console.log("File write completed. Executing drawGraph.js...");
    exec("node ./StateSpace/drawGraph.js", (error, stdout, stderr) => {
      if (error) {
        console.error(`Error executing drawGraph.js: ${error}`);
        return;
      }
      console.log(`drawGraph.js executed successfully.`);
    });
  }
});
fs.writeFile(
  "./UI/script.js",
  createScriptFile(forFileWrite(res["solution"]).join("").trim()),
  (err) => {
    if (err) {
      console.error(err);
      return;
    }
    console.log("./script.js file created successfully.");
    exec("start ./UI/index.html", (error) => {
```

```
        if (error) {
          console.error(`Error opening file: ${error}`);
        }
      });
    }
  );
  fs.writeFile(
    "./solution/sol.txt",
    forSolutionWrite(res["solution"]).join("").trim(),
    (err) => {
      if (err) {
        console.error(err);
      }
    }
  );
}
bfsSolutionAndDataSaver();
```

extractData.js

Processing data generated to represent it in graph-like structure.

```
const fs = require("fs");

function extractData(path) {
  const networkData = fs.readFileSync(path, "utf-8");

  const edges = networkData.split("\n").map((line) => {
    const [source, target, distance, color, fromLevel, toLevel] =
      line.split(" ");
    return {
      from: source,
      to: target,
      distance: distance,
      color: color,
      fromLevel: fromLevel,
      toLevel: toLevel,
    };
  });
  const nodes = [];
  edges.forEach((edge) => {
    if (!nodes.some((node) => node.id === edge.from)) {

      nodes.push({ id: edge.from, color: edge.color, level: edge.fromLevel });
    }
    if (!nodes.some((node) => node.id === edge.to)) {
      if(edge.to == JSON.stringify([3,2,0])){
```

```
        nodes.push({ id: edge.to, color: 3, level: edge.toLevel });
      }else if (edge.to == JSON.stringify([0,0,0])){
        nodes.push({ id: edge.to, color: 4, level: edge.toLevel });
      }else{
        nodes.push({ id: edge.to, color: edge.color, level: edge.toLevel });
      }
    }
  });
  return { edges: edges, nodes: nodes };
}

module.exports = {
  extractData,
};
```

createHtmlContent.js

Templating code to represents the problem in state space tree.

```
function createHtmlContent(nodes, edges, type) {
  const weightedEdges = edges.map((edge) => ({
    from: edge.from,
    to: edge.to,
    label: edge.distance.toString(),
  }));

  return `
    <!DOCTYPE html>
    <html>
    <head>
      <title>Graph Visualization</title>
       <style>
    #network-container {
      display: flex;
      width: 100%;
      height: 550px;
      border: 2px solid lightgray;
    }

    #nav-bar {
      width: 100%;
    }

    h1 {
      text-align: center;
    }
```

```
  h3 {
    display: block;
  }

  .legend {
    width: 100%;
    text-align: center;
    display: flex;
  }

  .red,
  .green,
  .yellow,
  .blue,
  .purple {
    width: 20%;
    color: white;
  }

  .red {
    background-color: red;
  }

  .green {
    background-color: green;
  }

  .yellow {
    background-color: yellow;
    color: black;
  }
  .blue {
    background-color: blue;
  }
  .purple {
    background-color: purple;
  }
</style>
  </head>
  <body>
  <div id="nav-bar">
  <h1>Cannibal Missionary Problem State Space Tree</h1>
  <div class="legend">
    <h2 class="red">Killed</h2>
```

```
      <h2 class="green">Explored</h2>
      <h2 class="yellow">Unexplored</h2>
      <h2 class="blue">Unexpandable</h2>
      <h2 class="purple">Goal State</h2>

    </div>
  </div>
      <div id="network-container">
        <div id="graph-container"></div>
      </div>
      <script src="https://unpkg.com/vis-network/standalone/umd/vis-
network.min.js"></script>
      <script>
        const container = document.getElementById('graph-container');
        const data = {
          nodes: ${JSON.stringify(
            nodes.map((node) => ({
              id: node.id,
              label: node.id,
              color:
                node.color == 1
                  ? "red"
                  : node.color == "0"
                  ? "green"
                  : node.color == "2"
                  ? "yellow"
                  : node.color == "3"
                  ? "blue"
                  : "purple",
              level: node.level,
            }))
          )},
          edges: ${JSON.stringify(weightedEdges)}
        };
        const options = {
      layout: {
        hierarchical: {
          direction: 'UD',
          sortMethod: 'directed',
          levelSeparation: 80,
          nodeSpacing: 150,
        }
      },
      physics: {
        enabled: false
```

```
      },
      nodes: {
        font: {
          size: 28,
          color: '#ffffff'
        }
      }
    };
        new vis.Network(container, data, options);
      </script>
    </body>
    </html>
  `;
}
module.exports = {
  createHtmlContent,
};
```

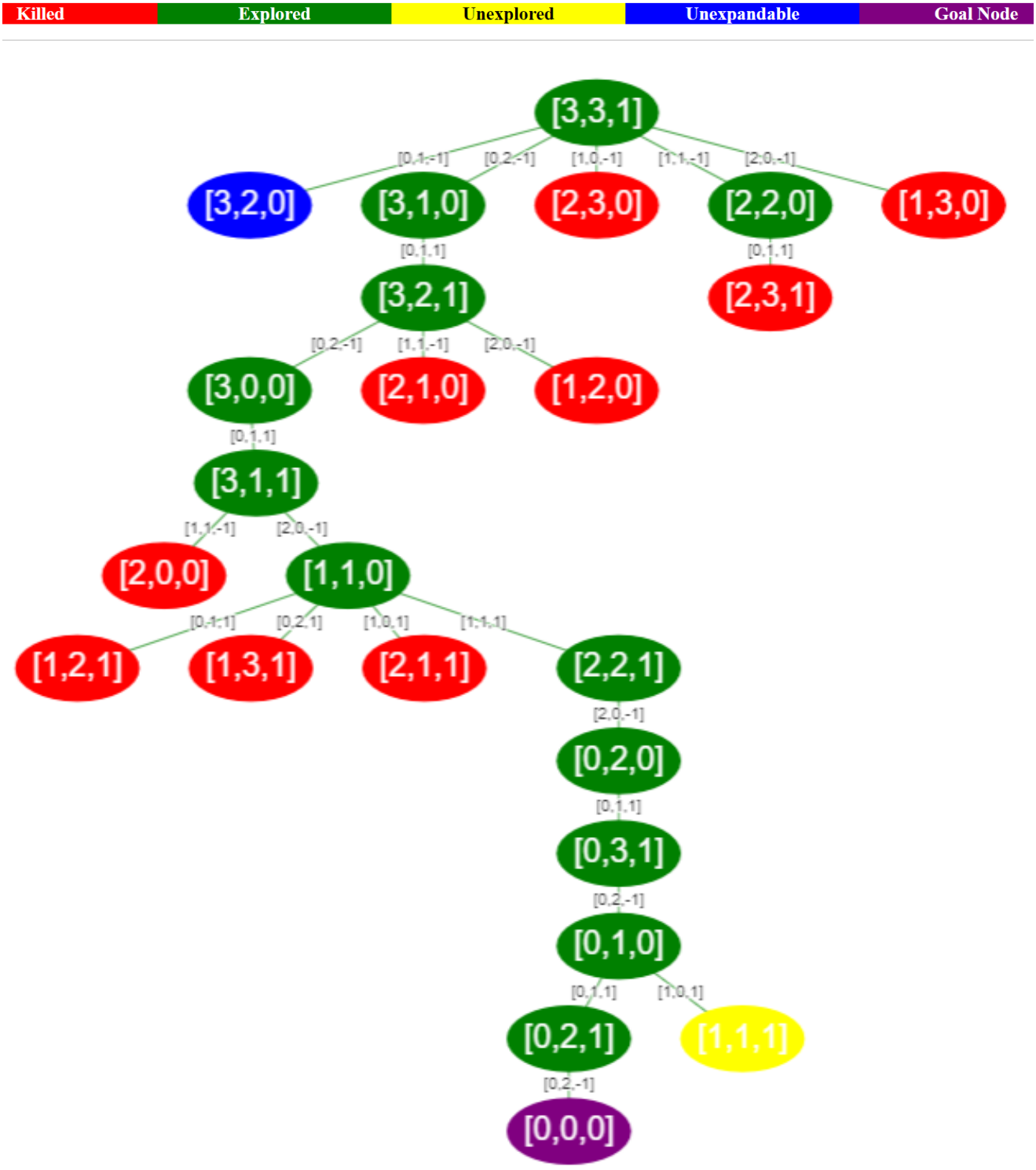
drawGraph.js

Following code trigger the extraction of data in desired format, and feed that data to template code and trigger the generation of final output i.e., state space tree.

```
const fs = require("fs");
const { extractData } = require("./extractData.js");
const { createHtmlContent } = require("./createHtmlContent.js");
const { exec } = require("child_process");
const data = extractData("./StateSpace/data.txt");
const nodes = data.nodes;
const edges = data.edges;
fs.writeFile(
  "./statespace.html",
  createHtmlContent(nodes, edges, "State Space Tree"),
  (err) => {
    if (err) {
      console.error(err);
      return;
    }
    console.log("./statespace.html file created successfully.");
    exec("start ./statespace.html", (error) => {
      if (error) {
        console.error(`Error opening file: ${error}`);
      }
    });
  }
);
```

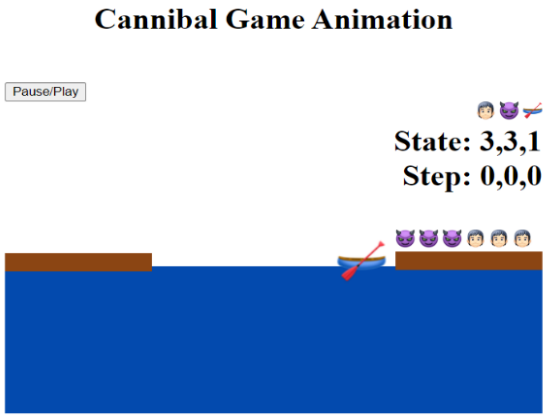
1.4 Output

Cannibal Missionary Problem State Space Tree

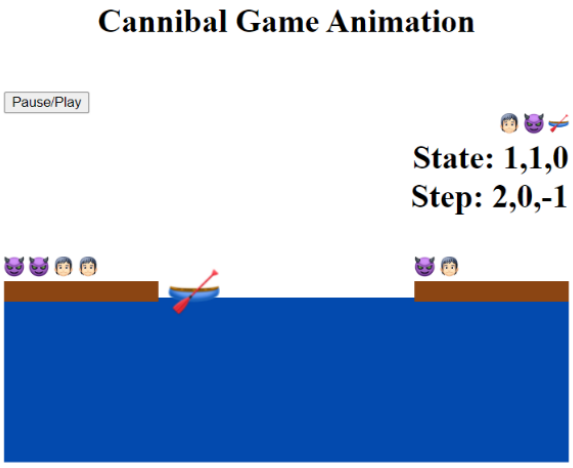


1.5 Animation:

Following screenshots shows some of transition state from one state to another and movement of missionary and cannible using simple animation.



Initial State:



Intermediate State:

Final State:

Q2. Explore the Solution of Q.No.1 using Breadth-first Search.

2.1 Introduction

In Breadth-first Search, shallowest unexpanded node is explored first. i.e., first-in-first-out (FIFO) queue, new successors go at end of the queue. This searching strategy is complete as it always reaches goal if the depth is finite. It gives optimal solutions if we guarantee that deeper solutions

are less optimal, by specifying some step-cost. In BFS, space is bigger problem in compare to time, as every node is kept in memory.
The following screenshots show the output of implementation of missionary and cannibal problem using BFS:

2.2 Source Code

fullBFS/main.js
Following code is the implementation of a breadth-first search (BFS) algorithm to solve a variation of the classic missionary and cannibal problem.

Breakdown of the code:

- **Node class:** Defining a class representing nodes in the problem space. It includes methods to check if a state is the goal state, if it's valid, generate child nodes, find a solution path, and validate the path to the goal.

- **BFS Algorithm (bfs):** Begins by creating a start node with an initial state. It explores nodes in a BFS manner, generating child nodes and examining their validity. It maintains a queue (**`q`)** of nodes to explore**.**

- **Node Expansion and Validation:** It iterates through the nodes in the queue, generating child nodes and validating their properties if a valid goal state is found, it checks if the path leading to that state is a valid solution path (**`isSolution`** method).

- **Depth Limit and Solution Handling:** There's a **`maxDepth`** constant that limits the depth of exploration. If the algorithm reaches this depth without finding a solution, it stops a return a message indicating the solution wasn't found.

- **Data Handling and Saving:** As the algorithm explores, it records the traversal tree information into arrays (**`from`, `to`, `weight`).** It also generates a solution path and saves the data into text files for further analysis.

fullBFS/main.js

```
const fs = require("fs");
let tree = [];
let from = [];
let to = [];
let weight = [];
const freezeTime = 100;
const maxDepth = 4;
function delay(time) {
```

```
  return new Promise((resolve) => setTimeout(resolve, time));
}
class Node {
  constructor(state, parent, action, depth) {
    this.parent = parent;
    this.state = state;
    this.action = action;
    this.depth = depth;
  }
  isGoal() {
    if (this.state[0] == 0 && this.state[1] == 0 && this.state[2] == 0) {
      return true;
    }
    return false;
  }
  isValid() {
    let missionaries = this.state[0];
    let cannibals = this.state[1];
    let boat = this.state[2];
    if (missionaries < 0 || missionaries > 3) {
      return false;
    }
    if (cannibals < 0 || cannibals > 3) {
      return false;
    }
    if (boat > 1 || boat < 0) {
      return false;
    }
    return true;
  }
  mustBeKilled() {
    let missionaries = this.state[0];
    let cannibals = this.state[1];
    if (missionaries < cannibals && missionaries > 0) {
      return true;
    }
    if (missionaries > cannibals && missionaries < 3) {
      return true;
    }
  }
  generateChild() {
    let children = [];
    let canoe = -1;
    if (this.state[2] == 0) {
      canoe = 1;
```

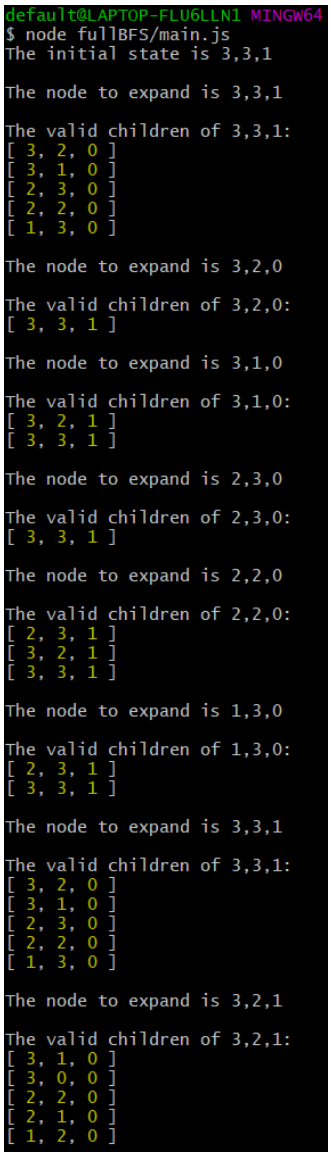
```
    }
    for (let x = 0; x < 3; x++) {
      for (let y = 0; y < 3; y++) {
        let newState = this.state.slice();
        newState[0] = newState[0] + canoe * x;
        newState[1] = newState[1] + canoe * y;
        newState[2] = newState[2] + canoe * 1;
        let action = [x, y, canoe];
        const newNode = new Node(newState, this, action, this.depth + 1);
        if (x + y >= 1 && x + y <= 2) {
          children.push(newNode);
        }
      }
    }
    return children;
  }
  findSolution() {
    let solution = [];
    solution.push(this.action);
    let path = new Node(this.state, this.parent, this.action);

    // Move up the tree
    while (path.parent != null) {
      path = path.parent;
      solution.push(path.action);
    }
    solution.pop();
    solution.reverse();
    return solution;
  }
  isSolution() {
    let solution = [];
    solution.push(this.action);
    let path = new Node(this.state, this.parent, this.action);
    while (path.parent != null) {
      path = path.parent;
      if (path.mustBeKilled()) {
        return false;
      }
    }
    return true;
  }
}
async function bfs(initialState) {
  let startNode = new Node(initialState, null, null, 0);
```

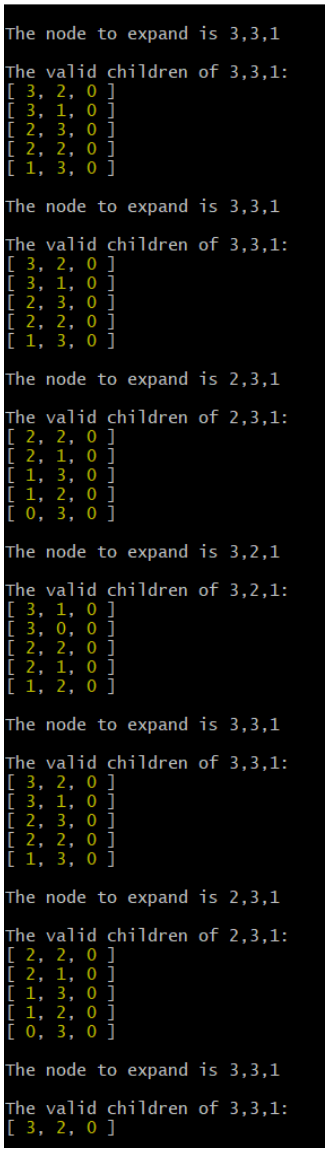
```
if (startNode.isGoal()) {
  return startNode.findSolution();
}
q = [];
q_states = [];
console.log(`The initial state is ${startNode.state}`);
q.push(startNode);
q_states.push(startNode.state);
while (!(q.length == 0)) {
  let node = q.shift();
  q_states.shift();

  console.log(`\nThe node to expand is ${node.state}\n`);
  let children = node.generateChild();
  console.log(`The valid children of ${node.state}:`);
  let count = 0;
  for (const child of children) {
    if (child.depth > maxDepth) {
      console.log(
        "\x1b[31m",
        `\n Max depth limit reached which was ${maxDepth}.`
      );
      return {
        solution: "Solution not found",
      };
    }
    if (child.isValid()) {
      count++;
      q.push(child);
      q_states.push(child.state);
      from.push([node.state, node.depth]);
      to.push([child.state, child.depth]);
      weight.push(child.action);
      console.log(child.state);
      if (child.isGoal()) {
        console.log("\nGoal State Found");
        console.log("Checking if it is our solution...");
        let isSolution = child.isSolution();
        if (!isSolution) {
          console.log(
            "\nThe path that leads to the goal state is not a valid solution\n"
          );
        } else {
          console.log(
            "\nThe path that leads to the goal state is a valid solution\n"
```

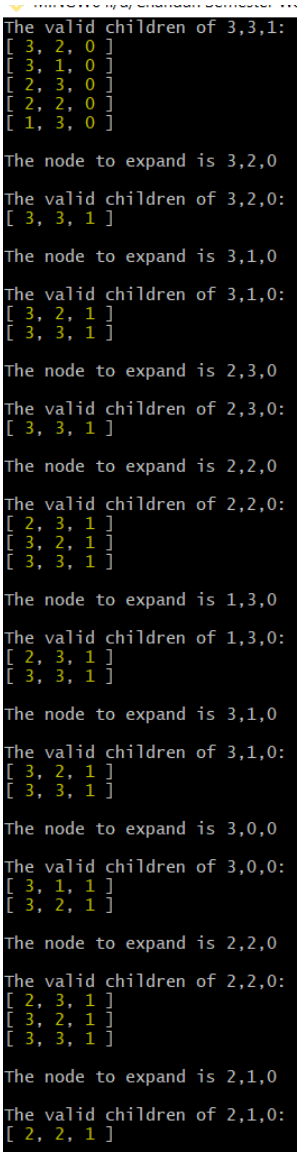
```
            );
            return {
              solution: child.findSolution(),
            };
          }
        }
      }
    }
    if (count === 0) {console.log("The node cannot be further expanded");}
  }
}
async function fullBfsSolutionAndDataSaver() {
  let initialState = [3, 3, 1];
  const res = await bfs(initialState);
  if (res["solution"] == "Solution not found") {
    console.log("\x1b[31m", res["solution"]);
    console.log("\x1b[37m");
  } else {
    console.log("Solution:");
    console.log(res["solution"]);
  }
  for (let i = 0; i < to.length; i++) {
    let appendData =
      JSON.stringify(from[i][0]) +" " +
      JSON.stringify(to[i][0]) +" " +
      JSON.stringify(weight[i]) +" " +"0" +" " +
      JSON.stringify(from[i][1]) +" " +
      JSON.stringify(to[i][1]) +"\n";
    tree.push(appendData);
  }
  function forSolutionWrite(arr) {
    let convertedData = [];
    for (data of arr) {
      let converted = JSON.stringify(data) + "\n";
      convertedData.push(converted);
    }
    return convertedData;
  }
  fs.writeFile("./fullBFS/data.txt", tree.join("").trim(), (err) => {
    if (err) console.error(err);
  });
  fs.writeFile(
    "./solution/solFullBFS.txt",
    res["solution"] == "Solution not found"
      ? "Solution not found"
```

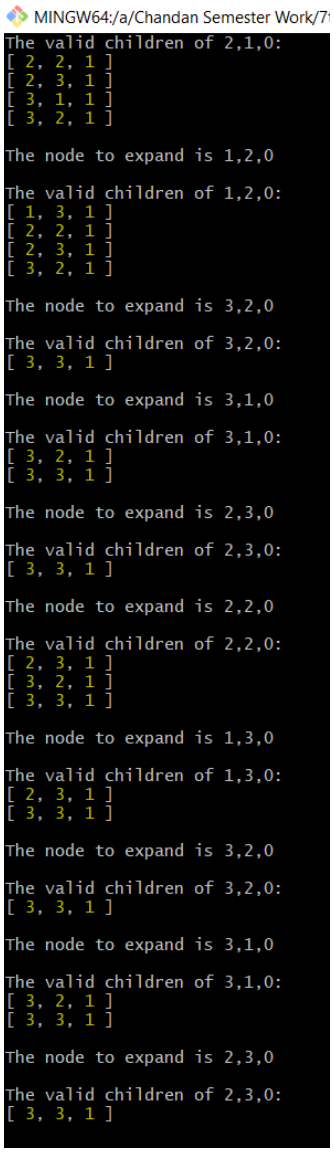
```
      : forSolutionWrite(res["solution"]).join("").trim(),
    (err) => {
      if (err) {
        console.error(err);
      }
    }
  );
}
fullBfsSolutionAndDataSaver();
```

2.3 Screenshots

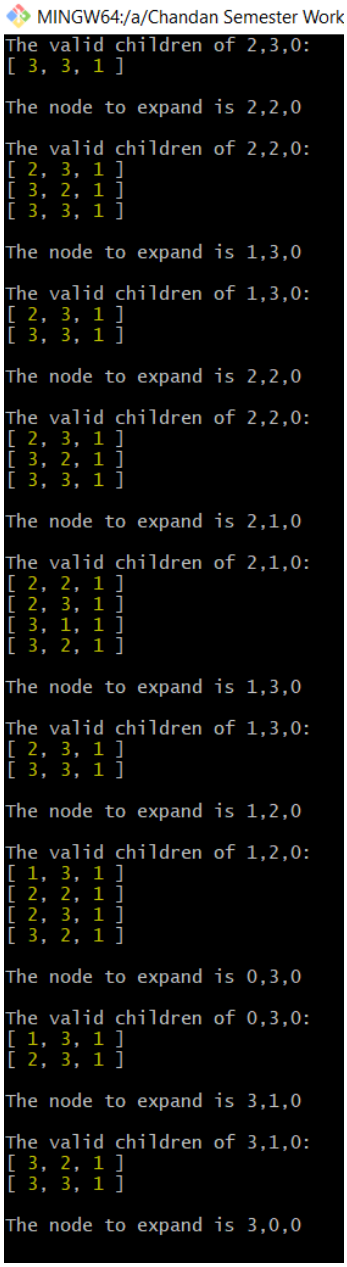


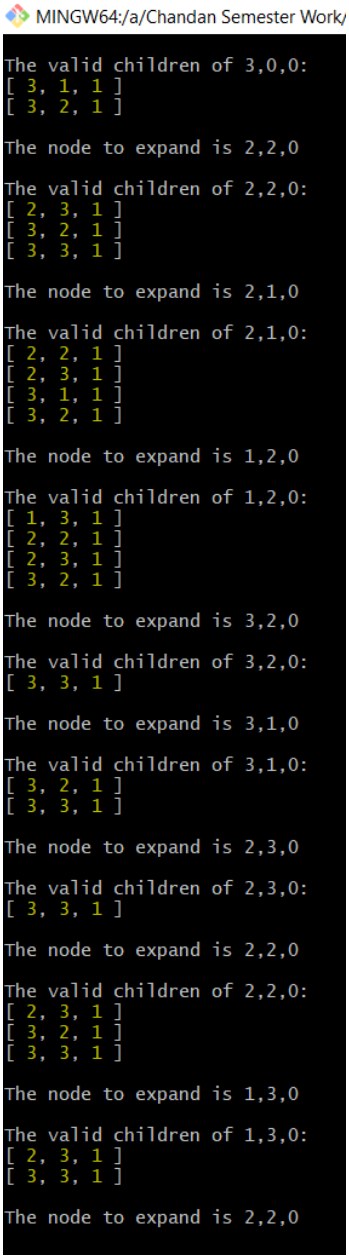


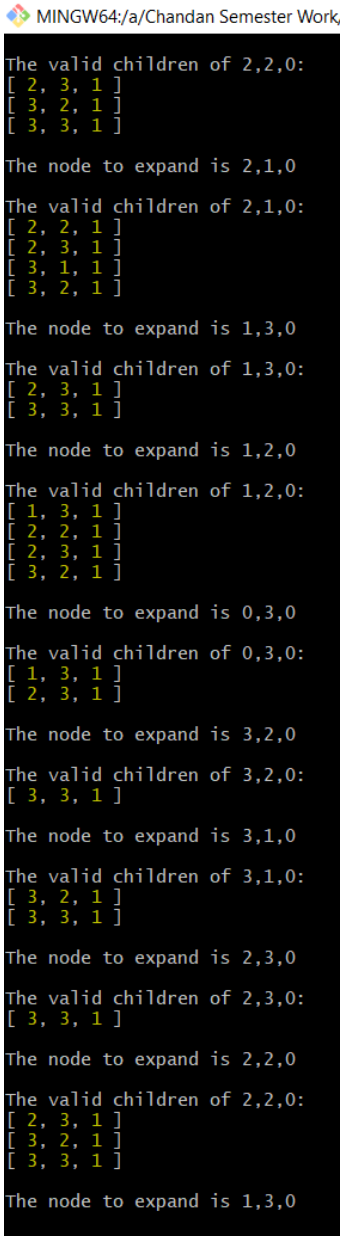


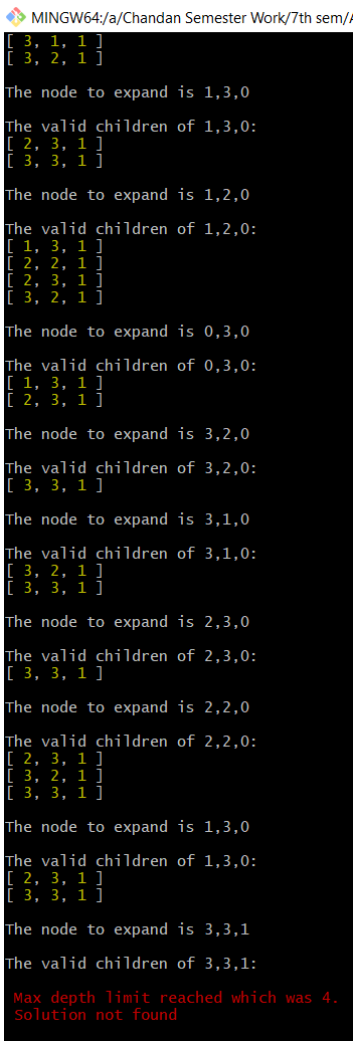


Screenshot 1 **Screenshot 2** **Screenshot 3** **Screenshot 4**









Screenshot 5 **Screenshot 6** **Screenshot 7** **Screenshot 8**

After exploring the nodes up to depth 4, as we have set the limit with depth 4 for the performance reason. Solution was not found, but from the question no 1 we know that if we continue the exploring of node up to depth 11, we can find the solution.

Q3. 8 Puzzle Game Using A* with heuristics (misplaced tiles)

Represent the state space for the 8 Puzzle Game using heuristics h1(n), number of misplaced tiles. Define the problem and generate the tree like structure which represent the state space.

3.1 Introduction

The 8-puzzle is a classic sliding puzzle game that consists of a 3x3 grid with eight numbered tiles and one missing tile. The goal of the game is to rearrange the tiles from an arbitrary starting configuration to a predefined goal configuration by sliding tiles into the empty space.

3.2 Game Components:

3.2.1 Configuration

The game is played on a 3x3 grid, and each cell can either be occupied by a numbered tile (1 to 8) or be empty. There are eight numbered tiles, usually arranged in random order at the beginning of the game. ne tile is left empty to allow the sliding of adjacent tiles.

3.2.2 Rules

The basic rule of the game is that a tile can be moved into the empty space if it is adjacent to the blank tile, either horizontally or vertically. Diagonal moves are not allowed. The game is considered solved when the tiles are arranged in the goal configuration.

3.3 Using A-Star

The A* algorithm is a pathfinding and graph traversal algorithm that efficiently finds the shortest path between two points on a graph. It is particularly popular in artificial intelligence, robotics, and computer games for task such as path planning. Also, A* incorporates heuristics to guide the search process.

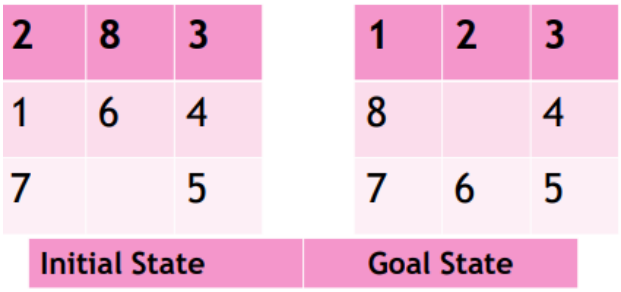
3.3.1 Problem Definition & State Representation

In the 8-puzzle game using missing tile, we define the initial state and goal state and move the tiles in all the possible way which produce multiple states, then we use the heuristics to determine which node to explore next. Since, we are using heuristics h1(n) = number of misplaced tiles. We will expand the node with lesser number of misplaced tiles at any given depth.

Mathematically,
f = g + h
where, g is depth of node
h is number of misplaced tiles.
The node with lesser value of f is explored first.

⇨ State Representation

A state of 8-puzzle game with missing tile can be represented as follow:



An operation takes us from one state to another, and this process is shown below using a tree like structure which has been generated programmatically from the implementation of 8-puzzle game using JavaScript, and graphing library Vis.js

3.4 Source Code

heuristics.js
Following code implements algorithm to solve the 8-puzzle game with a heuristic called "Misplaced Tiles".

Breakdown of heuristics.js:

1. **EightPuzzleState Class:** It represents the state of puzzle, important methods of this class are **isGoal(goalState)**, **getPossibleMoves()**, and **generateChild(move).**

2. **CalculateMisplacedTiles Function:** Calculates the number of tiles in the wrong position between the current state and the goal state.

3. **solveEightPuzzle Function:**

- Initializes a queue, a set of tracks visited states, and defines initial nodes for the puzzle.
- Calculate the heuristics value (number of misplaced tiles) for the initial state and adds it to the nodes array.

- Begins a while loop to perform BFS.
- Generates child nodes for the current node based on possible moves.
- Checks if the child state is not visited and adds it to the nodes and edges arrays.
- Select the best sibling node (based on the heuristic) to push into the queue for further exploration.
- Continue until a goal state is found or the queue is empty.

heuristics.js

```
class EightPuzzleState {
    constructor(state, parent = null, move = null, level = 0) {
        this.state = state;
        this.parent = parent;
        this.move = move;
        this.level = level;
    }
    toString() {
        return JSON.stringify(this.state);
    }
    isEqual(other) {
        return JSON.stringify(this.state) === JSON.stringify(other.state);
    }
    hash() {
        return JSON.stringify(this.state);
    }
    isGoal(goalState) {
        return JSON.stringify(this.state) === JSON.stringify(goalState);
    }
    getPossibleMoves() {
        const moves = [];
        for (let i = 0; i < 3; i++) {
            for (let j = 0; j < 3; j++) {
                if (this.state[i][j] === 0) {
                    if (j > 0) moves.push([0, -1]); // Move Left
                    if (i < 2) moves.push([1, 0]); // Move Down
                    if (i > 0) moves.push([-1, 0]); // Move Up
                    if (j < 2) moves.push([0, 1]); // Move Right
                }
            }
        }
        return moves;
    }
    generateChild(move) {
        const newState = this.state.map(row => [...row]);
        for (let i = 0; i < 3; i++) {
```

```
            for (let j = 0; j < 3; j++) {
                if (this.state[i][j] === 0) {
                    const [ni, nj] = [i + move[0], j + move[1]];
                    [newState[i][j], newState[ni][nj]] = [newState[ni][nj],
newState[i][j]];
                }
            }
        }
        return new EightPuzzleState(newState, this, move, this.level + 1);
    }
}
function calculateMisplacedTiles(currentState, goalState) {
    let misplaced = 0;
    for (let i = 0; i < 3; i++) {
        for (let j = 0; j < 3; j++) {
            if (currentState[i][j] !== goalState[i][j]) {
                misplaced++;
            }
        }
    }
    return misplaced;
}
function solveEightPuzzle(initialState, goalState) {
    const queue = [];
    const visited = new Set();

    const initialNode = new EightPuzzleState(initialState);
    queue.push(initialNode);
    visited.add(JSON.stringify(initialState));

    const nodes = [];
    const edges = [];
    const hValue = calculateMisplacedTiles(initialNode.state, goalState);
    nodes.push({
        color: 'green',
        id: JSON.stringify(initialNode.state),
        label: JSON.stringify(initialNode.state),
        level: initialNode.level,
        hValue: `g=${initialNode.level}, h=${hValue}, f=${initialNode.level +
hValue}`,
    });
    while (queue.length) {
        const currentNode = queue.shift();

        if (currentNode.isGoal(goalState)) {
```

```
            break;
        }
        const siblings = [];
        for (const move of currentNode.getPossibleMoves()) {
            const childNode = currentNode.generateChild(move);
            const childStateString = JSON.stringify(childNode.state);
            if (!visited.has(childStateString)) {
                nodes.push({
                    color: childNode.isGoal(goalState) ? 'green' : 'red',
                    id: childStateString,
                    label: childStateString,
                    level: childNode.level,
                    hValue: `g=${childNode.level},
h=${calculateMisplacedTiles(childNode.state, goalState)}, f=${
                        childNode.level +
calculateMisplacedTiles(childNode.state, goalState)
                    }`,
                });
                edges.push({
                    arrows: 'to',
                    from: JSON.stringify(currentNode.state),
                    to: childStateString,
                });
                visited.add(childStateString);
                siblings.push(childNode);
            }
        }
        if (siblings.length > 0) {
            const minSibling = siblings.reduce((minNode, node) => {
                const minMisplacedTiles = calculateMisplacedTiles(minNode.state,
goalState);
                const nodeMisplacedTiles = calculateMisplacedTiles(node.state,
goalState);
                return nodeMisplacedTiles < minMisplacedTiles ? node : minNode;
            });
            for (const nodeData of nodes) {
                if (nodeData.id === JSON.stringify(minSibling.state)) {
                    nodeData.color = 'green';
                }
                if (nodeData.id === JSON.stringify(goalState)) {
                    nodeData.color = 'green';
                }
            }
            queue.push(minSibling);
        }
```

```
    }
    return [nodes, edges];
}
module.exports = {
  solveEightPuzzle
};
```

main.js

Following code trigger the solveEightPuzzle to execute and passes the formatted data to templating code to generate state space tree.

```
const fs = require("fs");
const { exec } = require("child_process");
const { createHtmlContent } = require("./createHTMLContent");
const {solveEightPuzzle } = require("./heuristics");
function solve() {
    const initial_state = [
        [2, 8, 3],
        [1, 6, 4],
        [7, 0, 5]
    ];
    const goal_state = [
        [1, 2, 3],
        [8, 0, 4],
        [7, 6, 5]
    ];
    const [nodes, edges] = solveEightPuzzle(initial_state, goal_state);
    const graph_data = {
        nodes: nodes,
        edges: edges
    };
    return graph_data;
}
const graph_data = solve();
const nodesArray = [];
graph_data.nodes.forEach(function (data, index) {
    var dataarray = JSON.parse(data.label);
    var fillcolor = data.color
    var textcolor = 'black';
    var strokecolor = 'white';
    var strokewidth = 10;
    var fontsize = 75;
    var svgwidth = 400;
    var svgheight = 400;
    var rectwidth = 133.33;
```

```
    var rectheight = 133.33;
    var svg = '<svg width="' + svgwidth + '" height="' + svgheight + '"
xmlns="http://www.w3.org/2000/svg">';
    svg += '<rect x="0" y="0" width="' + rectwidth + '" height="' + rectheight +
'" fill="' + fillcolor + '" stroke="' + strokecolor + '" stroke-width="' +
strokewidth + '" />';
    svg += '<rect x="' + rectwidth + '" y="0" width="' + rectwidth + '" height="'
+ rectheight + '" fill="' + fillcolor + '" stroke="' + strokecolor + '" stroke-
width="' + strokewidth + '" />';
    svg += '<rect x="' + (2 * rectwidth) + '" y="0" width="' + rectwidth + '"
height="' + rectheight + '" fill="' + fillcolor + '" stroke="' + strokecolor + '"
stroke-width="' + strokewidth + '" />';
    svg += '<rect x="0" y="' + rectheight + '" width="' + rectwidth + '"
height="' + rectheight + '" fill="' + fillcolor + '" stroke="' + strokecolor + '"
stroke-width="' + strokewidth + '" />';
    svg += '<rect x="' + rectwidth + '" y="' + rectheight + '" width="' +
rectwidth + '" height="' + rectheight + '" fill="' + fillcolor + '" stroke="' +
strokecolor + '" stroke-width="' + strokewidth + '" />';
    svg += '<rect x="' + (2 * rectwidth) + '" y="' + rectheight + '" width="' +
rectwidth + '" height="' + rectheight + '" fill="' + fillcolor + '" stroke="' +
strokecolor + '" stroke-width="' + strokewidth + '" />';
    svg += '<rect x="0" y="' + (2 * rectheight) + '" width="' + rectwidth + '"
height="' + rectheight + '" fill="' + fillcolor + '" stroke="' + strokecolor + '"
stroke-width="' + strokewidth + '" />';
    svg += '<rect x="' + rectwidth + '" y="' + (2 * rectheight) + '" width="' +
rectwidth + '" height="' + rectheight + '" fill="' + fillcolor + '" stroke="' +
strokecolor + '" stroke-width="' + strokewidth + '" />';
    svg += '<rect x="' + (2 * rectwidth) + '" y="' + (2 * rectheight) + '"
width="' + rectwidth + '" height="' + rectheight + '" fill="' + fillcolor + '"
stroke="' + strokecolor + '" stroke-width="' + strokewidth + '" />';
    svg += '<text x="' + (rectwidth / 2) + '" y="' + (rectheight / 2) + '" font-
size="' + fontsize + '" text-anchor="middle" fill="' + textcolor + '">' +
dataarray[0][0] + '</text>';
    svg += '<text x="' + (rectwidth + (rectwidth / 2)) + '" y="' + (rectheight /
2) + '" font-size="' + fontsize + '" text-anchor="middle" fill="' + textcolor +
'">' + dataarray[0][1] + '</text>';
    svg += '<text x="' + ((2 * rectwidth) + (rectwidth / 2)) + '" y="' +
(rectheight / 2) + '" font-size="' + fontsize + '" text-anchor="middle" fill="' +
textcolor + '">' + dataarray[0][2] + '</text>';
    svg += '<text x="' + (rectwidth / 2) + '" y="' + (rectheight + (rectheight /
2)) + '" font-size="' + fontsize + '" text-anchor="middle" fill="' + textcolor +
'">' + dataarray[1][0] + '</text>';
    svg += '<text x="' + (rectwidth + (rectwidth / 2)) + '" y="' + (rectheight +
(rectheight / 2)) + '" font-size="' + fontsize + '" text-anchor="middle" fill="'
+ textcolor + '">' + dataarray[1][1] + '</text>';
```

```
    svg += '<text x="' + ((2 * rectwidth) + (rectwidth / 2)) + '" y="' +
(rectheight + (rectheight / 2)) + '" font-size="' + fontsize + '" text-
anchor="middle" fill="' + textcolor + '">' + dataarray[1][2] + '</text>';
    svg += '<text x="' + (rectwidth / 2) + '" y="' + ((2 * rectheight) +
(rectheight / 2)) + '" font-size="' + fontsize + '" text-anchor="middle" fill="'
+ textcolor + '">' + dataarray[2][0] + '</text>';
    svg += '<text x="' + (rectwidth + (rectwidth / 2)) + '" y="' + ((2 *
rectheight) + (rectheight / 2)) + '" font-size="' + fontsize + '" text-
anchor="middle" fill="' + textcolor + '">' + dataarray[2][1] + '</text>';
    svg += '<text x="' + ((2 * rectwidth) + (rectwidth / 2)) + '" y="' + ((2 *
rectheight) + (rectheight / 2)) + '" font-size="' + fontsize + '" text-
anchor="middle" fill="' + textcolor + '">' + dataarray[2][2] + '</text>';
    svg += '</svg>';
    nodesArray.push({
        id: data.id,
        shape: 'image',
        image: '' + btoa(svg),
        level: data.level,
        label: data.hValue
    });
});
var data = {
    nodes: nodesArray,
    edges: graph_data.edges,
};
fs.writeFile(
  "./Heuristics/heuristics.html",
  createHtmlContent(data),
  (err) => {
    if (err) {
      console.error(err);
      return;
    }
    console.log("./heuristics.html file created successfully.");
    exec("start ./Heuristics/heuristics.html", (error) => {
      if (error) {
        console.error(`Error opening file: ${error}`);
      }
    });
  }
);
```

createHTMLContent.js

```
function createHtmlContent(data){
    return `<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>State Space Tree Visualization</title>
    <script type="text/javascript" src="https://unpkg.com/vis-
network@7.7.0/dist/vis-network.min.js"></script>
     <style>
        html,
        body {
            height: 100%;
            margin: 0;
            overflow: hidden;
        }
        #mctree {
            width: 100%;
            height: 100vh;
        }
        .legend {
            position: fixed;
            left: 40%;
            margin-bottom: 60px;
        }
        .legend-item {
            display: flex;
        }
        .circle {
            width: 30px;
            height: 30px;
            border-radius: 50%;
            margin-right: 20px;
        }
        .legend-text {
            margin: 0;
            font-size: 30px;
        }
        #header {
            text-align: center;
            background: #fefefe;
            color: #000000;
            margin: 0;
        }
```

```
    </style>
</head>
<body>
     <div id="header">
        <h1 class="top-text">8 Puzzle Game State Space Tree Using Missing
Element.</h1>
        <div class="legend">
            <div class="legend-item">
                <div class="circle" style="background-color: green;"></div>
                <p class="legend-text">Solution Path</p>
            </div>
            <div class="legend-item">
                <div class="circle" style="background-color: red;"></div>
                <p class="legend-text">Killed Node</p>
            </div>
        </div>
    </div>
    <div id="mctree"></div>
    <script type="text/javascript" src="https://unpkg.com/vis-
network@7.7.0/dist/vis-network.min.js"></script>
    <script>
        var nodes = null;
        var edges = null;
        var network = null;
        var data = ${ JSON.stringify(data, null, 2)};
        var container = document.getElementById('mctree');
        var options = {
            layout: {
                hierarchical: {
                    direction: 'UD',
                    sortMethod: 'directed',
                    levelSeparation: 90,
                    nodeSpacing: 180,
                }
            },
            physics: {enabled: false},
        };
        var network = new vis.Network(container, data, options);
    </script>
</body>
</html>`
}
module.exports = {
    createHtmlContent
}
```

3.5 Output

8 Puzzle Game State Space Tree Using Missing Element.