Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра АСОІУ

ЗВІТ

про виконання лабораторної роботи № 1

з дисципліни

«Основи штучного інтелекту»

Варіант 4

|  |  |
| --- | --- |
| Перевірила:  Ст. вик. Мажара О.О. | Виконала:  Студентка групи ІС-71  Вознюк Олександра |
|  |  |

Київ 2020

1. **Мета роботи**

Ознайомитися з методами неінформативного пошуку та розробити алгоритм на їх основі для вирішення задачі згідно варіанту.

1. **Завдання до роботи**

Розробити алгоритм вирішення старовинної логічної задачі: "Як за допомогою 5-ти літрового і 9-ти літрового відра набрати з річки 3 літри води?"

1. **Текст розробленого програмного забезпечення з коментарями.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AI1

{

class Program

{

static void Main(string[] args)

{

Node root = new Node(1, 0, 0, null);

DFS dfs = new DFS();

dfs.Search(root);

Console.WriteLine("Depth Search with limit");

Console.WriteLine("Jar1 = 9 Jar2=5 Limit=20");

Console.WriteLine("Path:");

dfs.GetPath();

Console.WriteLine();

Console.WriteLine("All visited vertexes:");

foreach (var n in dfs.visited)

Console.WriteLine(n);

Console.WriteLine();

//dfs.printTree(root);

Console.ReadLine();

}

}

}

using System.Collections.Generic;

namespace AI1

{

public class Node

{

public int Level { get; set; }

//-----------------------------------

public int Jar1 { get; set; }

public int Jar2 { get; set; }

//----------------------------------------------

public static int limitFirstJar = 9;

public static int limitSecondJar = 5;

//----------------------------------------------------

public static int levelLimit = 20;

public Node ParentNode { get; set; }

public List<Node> \_children = new List<Node>();

public Node(int level, int jar1, int jar2, Node parent)

{

Level = level;

Jar1 = jar1;

Jar2 = jar2;

ParentNode = parent;

//MakeChildren();

}

public void MakeChildren()

{

if (Level <= levelLimit)//tree has 20 or less levels

{

if (Jar2 == 0 && Jar1 != limitFirstJar)//full jar2

{

\_children.Add(new Node(this.Level + 1, Jar1, limitSecondJar, this));

}

if (Jar1 == 0 && Jar2 != limitSecondJar)//full jar1

{

\_children.Add(new Node(this.Level + 1, limitFirstJar, Jar2, this));

}

if (Jar1 != limitFirstJar && Jar2 != 0)//from jar2 to jar1

{

if (Jar2 >= limitFirstJar - Jar1)//jar2 has more water than jar1 can contain

{

\_children.Add(new Node(this.Level + 1, limitFirstJar, Jar2 - (limitFirstJar - Jar1), this));

}

else

{

\_children.Add(new Node(this.Level + 1, Jar2 + Jar1, 0, this));

}

}

if (Jar2 != limitSecondJar && Jar1 != 0)//from jar1 to jar2

{

if (Jar1 >= limitSecondJar - Jar2)//jar1 has more water than jar1 can contain

{

\_children.Add(new Node(this.Level + 1, Jar1 - (limitSecondJar - Jar2), limitSecondJar, this));

}

else

{

\_children.Add(new Node(this.Level + 1, 0, Jar1 + Jar2, this));

}

}

if (Jar1 != 0 && Jar2 != 0)//empty jar1

{

\_children.Add(new Node(this.Level + 1, 0, Jar2, this));

}

if (Jar1 != 0 && Jar2 != 0)//empty jar2

{

\_children.Add(new Node(this.Level + 1, Jar1, 0, this));

}

}

}

public override string ToString()

{

string node = "Jar1: " + Jar1.ToString() + " " + "Jar2: " + Jar2.ToString() + " Level: " + Level + "\n";

if (ParentNode != null)

{

string parentNode = " Parent: " + ParentNode.Jar1 + " " + ParentNode.Jar2 + " Level: " + ParentNode.Level + "\n";

return node + parentNode;

}

else

return node;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AI1

{

public class DFS

{

private List<Node> path = new List<Node>();

public List<Node> visited = new List<Node>();

public static int result = 3;

public static int depthLimit = 30;

public DFS()

{

}

public void Search(Node root)

{

root.MakeChildren();//generate possible states

foreach (var node in root.\_children)//for each state

{

if (!IsResult())//if result is not found

{

visited.Add(root);//add to visited nodes

if (!PathHasNode(node) && visited.Count<=depthLimit)//if path don't have this node

{

path.Add(root);//add to path

Search(node);//continue recursion with children node

}

}

}

}

public bool IsResult()//check whether result is in the path or no

{

foreach(var node in path)

{

if (node.Jar1 == result || node.Jar2 == result)

{

return true;

}

}

return false;

}

public void GetPath()//print path

{

foreach(var node in path)

{

Console.WriteLine(node);

}

}

public bool PathHasNode(Node n)//check if a concrete node is present in the path

{

foreach (var node in path)

{

if (node.Jar1 == n.Jar1 && node.Jar2 == n.Jar2)

{

return true;

}

}

return false;

}

public void printTree(Node root)

{

foreach(var node in root.\_children)

{

Console.WriteLine(node);

Console.WriteLine();

printTree(node);

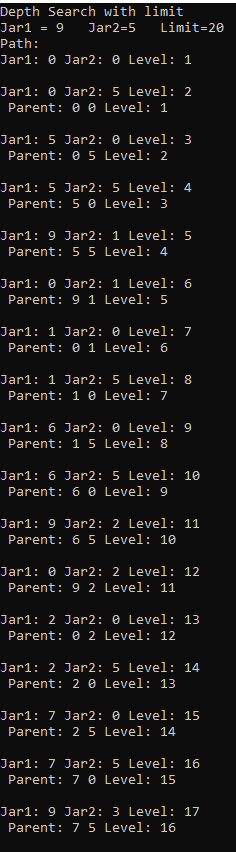
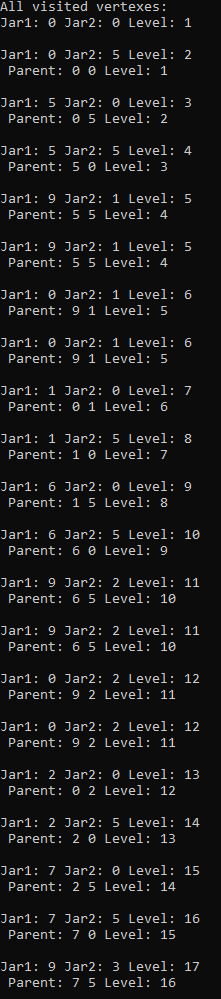
}

}

}

}

1. **Результати роботи програмного забезпечення, що включають результати тестування та копії екранних форм.**

1. **Висновки, що відображають особисто отримані результати виконання роботи, їх критичний аналіз.**

Метод пошуку в глибину з обмеженням на глибину не дає оптимального результату, якщо поставити низький ліміт. Якщо не контролювати виділення пам’яті для породження наступних станів може статися її переповнення. Особисто отримані мною результати показують, що метод призводить до правильного рішення. Але в отриманих мною результатах ви можете побачити що в списку усіх відвіданих вершин деякі вершини повторюються. Думаю, це потрібно розуміти як те, що якщо пошук в глибину потрапляє в стан, що вже є в списку шляху, то ми починаємо рухатися горизонтально по одному рівню перебираючи дітей доки не знайдемо підходящий вузол або завершимо роботу.