Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

«Київський політехнічний інститут»

Кафедра АСОІУ

ЗВІТ

про виконання лабораторної роботи № 3

з дисципліни

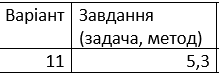
«[Основи штучного інтелекту](http://wiki.kpi.ua/index.php/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%B8%20%D1%88%D1%82%D1%83%D1%87%D0%BD%D0%BE%D0%B3%D0%BE%20%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82%D1%83_(20202020))»

|  |  |
| --- | --- |
| Перевірив:  [ст.вик. Мажара О. О.](http://rozklad.kpi.ua/Schedules/ViewSchedule.aspx?v=49d45d55-bbb9-49a2-a042-8befd9bb0ab1) | Виконав:  Студент групи ІС-73  Коноплянка Д. С. |
|  |  |

Київ 2020

**Завдання до роботи.**

Варіант 11



Сталося так, що до берега великого Гангу під'їхали відразу троє магараджею зі своїми пери. Всі вони хотіли переправитися на інший берег, але звичаї не дозволяли жодній пери залишатися в човні або на березі одній з чужим чоловіком, якщо поруч не буде свого. Біля берега стояв невеликий човен, який витримує тільки двох людей. Звичаї не забороняють пери самої керувати човном. Потрібно знайти послідовність поїздок, яка гарантує переміщення всіх магараджі і пери на інший берег.

**Текст розробленого програмного забезпечення з коментарями.**

class Node

{

public Node Prev { set; get; }

public string Actionlanded { set; get; }

public string ActionBoad { set; get; }

public string ActionStayed { set; get; }

public Shore FirstShore { set; get; }

public Shore SecondShore { set; get; }

public int Value { set; get; }

public List<Node> ExpandNodes()

{

List<Node> childs = new List<Node>();

foreach (var item in FirstShore.Persons)

{

foreach (var item2 in FirstShore.Persons)

{

if (FirstShore.Persons.Count == 1)

{

var firstNode = GenerateMove(FirstShore, SecondShore, item, this, new Person() { Pair = 0, Type = "" });

if (firstNode != null)

{

childs.Add(firstNode);

}

}

else

{

if (item.Type == "Magaradja")

{

if (item2.Type == "Magaradja")

{

if (item.Pair != item2.Pair)

{

var firstNode = GenerateMove(FirstShore, SecondShore, item, this, item2);

var secondNode = GenerateMove(FirstShore, SecondShore, item2, this, item);

if (firstNode != null)

{

childs.Add(firstNode);

}

if (secondNode != null)

{

childs.Add(secondNode);

}

}

}

else if (item2.Type == "Peri")

{

if (item.Pair == item2.Pair)

{

var firstNode = GenerateMove(FirstShore, SecondShore, item, this, item2);

var secondNode = GenerateMove(FirstShore, SecondShore, item2, this, item);

if (firstNode != null)

{

childs.Add(firstNode);

}

if (secondNode != null)

{

childs.Add(secondNode);

}

}

}

}

if (item.Type == "Peri")

{

if (item2.Type == "Magaradja")

{

if (item.Pair == item2.Pair)

{

var firstNode = GenerateMove(FirstShore, SecondShore, item, this, item2);

var secondNode = GenerateMove(FirstShore, SecondShore, item2, this, item);

if (firstNode != null)

{

childs.Add(firstNode);

}

if (secondNode != null)

{

childs.Add(secondNode);

}

}

}

if (item2.Type == "Peri")

{

if (item.Pair != item2.Pair)

{

var firstNode = GenerateMove(FirstShore, SecondShore, item, this, item2);

var secondNode = GenerateMove(FirstShore, SecondShore, item2, this, item);

if (firstNode != null)

{

childs.Add(firstNode);

}

if (secondNode != null)

{

childs.Add(secondNode);

}

}

}

}

}

}

}

return childs;

}

public Node GenerateMove(Shore FirstShore, Shore SecondShore, Person person, Node prev, Person secondPerson)

{

if (person.Type == "Peri")

{

var magaradji = SecondShore.Persons.FirstOrDefault(x => x.Pair == person.Pair && x.Type == "Magaradja");

if (magaradji == null)

{

return null;

}

}

var firstShore = new Shore();

var secondShore = new Shore();

var firstShoreFirstNode = FirstShore.Persons.Where(x => x != person).ToList();

var secondShoreFirstNode = SecondShore.Persons.ToList();

secondShoreFirstNode.Add(person);

firstShore.Persons = firstShoreFirstNode;

secondShore.Persons = secondShoreFirstNode;

var actionBoad = $"In Boad: {person.Type}, Pair: {person.Pair} and {secondPerson.Type}, Pair: {secondPerson.Pair}";

var actionStayed = $"Person: {secondPerson.Type}, Pair: {secondPerson.Pair} return boad to first shore";

var action = $"Person: {person.Type}, Pair: {person.Pair} move to second shore";

return new Node() { Prev = prev, FirstShore = firstShore, SecondShore = secondShore, Actionlanded = action, ActionBoad = actionBoad, ActionStayed = actionStayed, Value = GenerateValue(prev, person, secondPerson) };

}

static int GenerateValue(Node node, Person person, Person secondPerson)

{

if ((person.Pair == secondPerson.Pair) && (person.Type != secondPerson.Type))

{

return node.Value + 1;

}

else if ((person.Pair == secondPerson.Pair) || (person.Type != secondPerson.Type))

{

return node.Value + 2;

}

return node.Value + 3;

}

}

class Person

{

public int Pair { set; get; }

public string Type { set; get; }

}

class Shore

{

public List<Person> Persons { set; get; } = new List<Person>();

}

class Program

{

static void Main(string[] args)

{

var firstShore = new Shore();

var secondShore = new Shore();

var firstPeri = new Person() { Pair = 1, Type = "Peri" };

var secondPeri = new Person() { Pair = 2, Type = "Peri" };

var thirdPeri = new Person() { Pair = 3, Type = "Peri" };

var firstMagaradja = new Person() { Pair = 1, Type = "Magaradja" };

var secondMagaradja = new Person() { Pair = 2, Type = "Magaradja" };

var thirdMagaradja = new Person() { Pair = 3, Type = "Magaradja" };

firstShore.Persons = new List<Person>() { firstPeri, secondPeri, thirdPeri, firstMagaradja, secondMagaradja, thirdMagaradja };

var node = new Node()

{

FirstShore = firstShore,

SecondShore = secondShore

};

var sol = Algoritm(node);

if (sol != null)

{

Print(sol);

}

}

static void Print(Node node)

{

if (node.Prev != null)

{

Print(node.Prev);

Console.WriteLine(node.ActionBoad);

Console.WriteLine(node.ActionStayed);

Console.WriteLine(node.Actionlanded);

Console.WriteLine(new string('-', 50));

}

}

static Node Algoritm(Node node)

{

if (IsGoal(node))

{

return node;

}

var succesors = node.ExpandNodes();

if (succesors.Count == 0)

{

return null;

}

var minValue = succesors.Min(x => x.Value);

succesors = succesors.Where(x => x.Value == minValue).ToList();

foreach (var item in succesors)

{

var sr = Algoritm(item);

if (sr != null)

{

return sr;

}

}

return null;

}

static bool IsGoal(Node node)

{

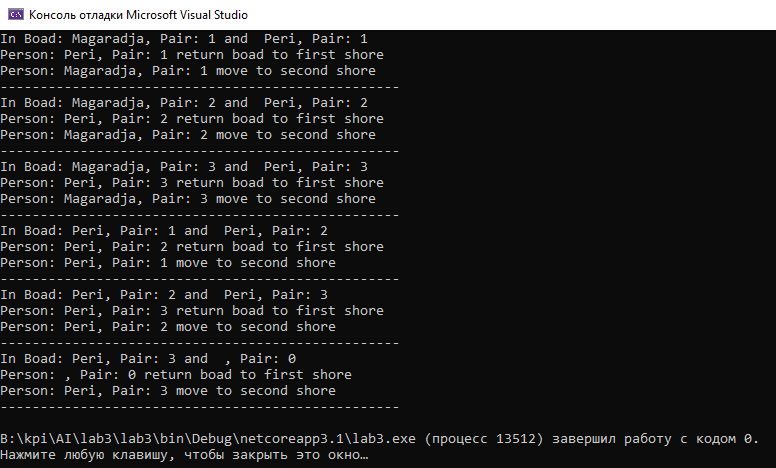
var check = node.SecondShore.Persons.Count == 6;

return check;

}

}

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



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

В даній роботі був використаний beam search - це евристичний алгоритм пошуку, який досліджує граф шляхом розширення найбільш перспективного вузла в обмеженому наборі. Пошук у променях - це оптимізація найкращого пошуку, що зменшує його потреби в пам'яті. Найкращий пошук - це графічний пошук, який упорядковує всі часткові рішення (стани) згідно з деякими евристичними. Але в пошуку променів лише кандидатами зберігається заздалегідь визначена кількість найкращих часткових рішень.

## Контрольні запитання

1. Яка відмінність між алгоритмами звичайного пошуку та локального пошуку?

В алгоритмах локального пошуку ведеться тільки на підставі поточного стану, а раніше пройдені стану не враховуються і не запам'ятовуються. Основною метою пошуку є не знаходження оптимального шляху до цільової точці, а оптимізація деякої цільової функції, тому завдання, які вирішуються подібними алгоритмами, називають завданнями оптимізації.

1. Поясніть в чому полягає складність роботи алгоритмів локального пошуку (пов’язана з ландшафтом станів)

Складність виявляється в опису простору станів в таких завданнях використовують ландшафт простору станів, в цьому поданні завдання зводиться до пошуку стану глобального максимуму (або мінімуму) на даному ландшафті.

1. Що таке задачі з обмеженнями (CSP)? Як вони сформульовані?

задачі з обмеженнями (УО) (constraint satisfaction problem). Метою рішення задачі УО є знаходження значень змінних, які відповідають заданим обмеженням. Проблема існування рішень задачі УО є NP-повною.

1. Поясніть принцип роботи алгоритмів, які реалізовані у лабораторній роботі.

В даній роботі був використаний beam search - це евристичний алгоритм пошуку, який досліджує граф шляхом розширення найбільш перспективного вузла в обмеженому наборі.