РУСЕНСКИ УНИВЕРСИТЕТ „АНГЕЛ КЪНЧЕВ”

**Курсова работа**

*по Изкуствен интелект*

На **Андриян Невелинов Кръстев**

Фак.номер: **1132250**

Група: **26б**

Специалност: **КСТ**

гр.Русе

2014 г.

**1.Задание № 2** :

**“Мисионери и канибали” (Missionary and cannibal problem) - I-ви вариант(с изполване на BFS):**

***N*** *мисионери (****N****>=1) и* ***N*** *канибали се намират на левия бряг на една река. Целта на задачата е всички мисионери и канибали да бъдат прехвърлени на десния бряг на реката като се използва лодка. Ограниченията са следните:*

*1. Лодката е най-много за двама човека (единият, от които гребе).*

*2. На кой да е бряг броят на мисионерите трябва да е по-голям или равен на броя на канибалите.*

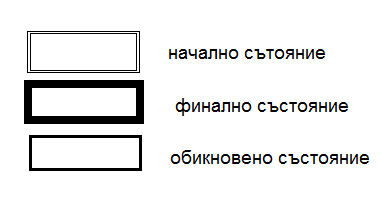
*Нека разходите за превозването на канибал са 1 единица, а на мисионер - 2 единици.*

**2. Представяне на задачата като задача за търсене в ПС („легенда” за разчитане на графа на ПС).**

Състоянията на задачата съм предватавил чрез наредена тройка (а,а1,а2), където а показва броя на канибалите и мисионерите на левия бряг а а1 тези на десния.Съответно а2 показва от коя страна се намира лодката. С m съм обозначил мисионерите, с с канибалите, L и R положението на лодката, следователно началното ми състояние е (nMnC/0,0 L) а крайното (0,0/nMnC R).Възможните преходи между две състояния са:

* 1m - прехвърляне на един мисионер;
* 1c - прехвърляне на един канибал;
* 2m - прехвърляне на два мисионера;
* 2c - прехвърляне на два канибала
* 1m1c - прехвърляне на един мисионер и един канибал;

ПС на тази задача е N=2 представено като граф.



Разходите за стрелките са:

* 1m – 2 единици;
* 1c – 1 единица;
* 2m – 4 единици;
* 2c – 2 единици;
* 1m1c – 3 единици;

**3. Граф на ПС.**



**4. Програмна реализация.**

4.1 Псевдо код на алгоритъма BFS(търсене в дълбочина)

method BFS

създаване на опашка Open;

създаване на лист Closed;

while (съдържанието на Open e различно от 0)

begin

Вадим най лявото състояние от Оpen и го поставяме в X;

if (X == Goal)

{

return X;

}

if (X се съдържа в Close)

{

Премини на следваща итерация

}

while(Генериране на наследници на Х)

определяме следващия наследник

if (проверяваме дали наследника отговаря на изискванията)

{

Добавя ме го в дясната страна на Open

}

Поставяме Х в Closed;

return null;

}

4.2 Модули

BsfAlgorithm.cs

Имплементация на BFS алгоритъмът.Основният метод е BFS. В този клас са имплементирани и методите:

IsRepeat- дали се поваря дадено състояние

IsContentInOpen – дали в Open се съдържа текущото състояние

CheckIsValidNode – дали текущото състояние е валидно

CheckGoalNode – дали е крайно състояние

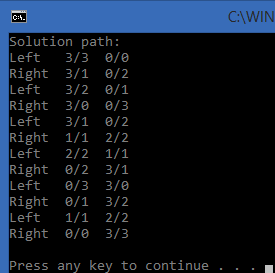
State.cs

Създаване на клас State.Дефиниране на конструктор и override на метода ToString().

SolutionTest.cs

Тестване на задачата.

4.3 Тестови примери.



5.4 Разпечатка на кода на разработения софтуер.

namespace MissionaryProblem

{

using System;

using System.Collections;

using System.Collections.Generic;

class BfsAlgorithm

{

public static State BFS(State start)

{

//List<State> allSolution = new List<State>();

Queue<State> Open = new Queue<State>();

List<State> Closed = new List<State>();

Open.Enqueue(start);

while (Open.Count > 0)

{

State X = Open.Dequeue();

if (CheckIfGoalNode(X))

{

//allSolution.Add(X);

//if (Open.Count != 0)

//{

// continue;

//}

//else

//{

return X;

//}

}

if (IsRepeat(Closed, X))

{

continue;

}

for (int i = 0; i < 5; i++)

{

int[] arr = null;

if (X.boatSide)

{

switch (i)

{

case 0: arr = new int[] { -1, 0, 1, 0 }; break; //One missionar

case 1: arr = new int[] { 0, -1, 0, 1 }; break; //One cannibal

case 2: arr = new int[] { -2, 0, 2, 0 }; break; //Two missionary

case 3: arr = new int[] { 0, -2, 0, 2 }; break; //Two cannibals

case 4: arr = new int[] { -1, -1, 1, 1 }; break; //One missionar and one cannibal

}

}

else

{

switch (i)

{

//same like above but for another bank of river

case 0: arr = new int[] { 1, 0, -1, 0 }; break;

case 1: arr = new int[] { 0, 1, 0, -1 }; break;

case 2: arr = new int[] { 2, 0, -2, 0 }; break;

case 3: arr = new int[] { 0, 2, 0, -2 }; break;

case 4: arr = new int[] { 1, 1, -1, -1 }; break;

}

}

State next = new State(X.numberMissionaryLeft + arr[0], X.numberCannibalLeft + arr[1], X.numberMissionaryRight + arr[2], X.numberCannibalRight + arr[3], !X.boatSide, X);

if (CheckIfValidNode(next))

{

Open.Enqueue(next);

}

}

//if (!IsContentInOpen(Open, X))

//{

Closed.Add(X);

//}

}

return null;

}

public static bool IsRepeat(List<State> list,State X)

{

foreach (var item in list)

{

if (item.numberCannibalLeft == X.numberCannibalLeft && item.numberCannibalRight == X.numberCannibalRight && item.numberMissionaryLeft == X.numberMissionaryLeft && item.numberMissionaryRight == X.numberMissionaryRight && item.boatSide == X.boatSide)

{

return true;

}

}

return false;

}

public static bool IsContentInOpen(Queue<State> list, State X)

{

foreach (var item in list)

{

if (item.numberCannibalLeft == X.numberCannibalLeft && item.numberCannibalRight == X.numberCannibalRight && item.numberMissionaryLeft == X.numberMissionaryLeft && item.numberMissionaryRight == X.numberMissionaryRight && item.boatSide == X.boatSide)

{

return true;

}

}

return false;

}

public static bool CheckIfValidNode(State X)

{

if ((X.numberMissionaryLeft > 0 && X.numberCannibalLeft > X.numberMissionaryLeft) || (X.numberMissionaryRight > 0 && X.numberCannibalRight > X.numberMissionaryRight))

{

return false;

}

if (X.numberCannibalLeft < 0 || X.numberMissionaryLeft < 0 || X.numberCannibalRight < 0 || X.numberMissionaryRight < 0)

{

return false;

}

return true;

}

public static bool CheckIfGoalNode(State X)

{

if (X.numberMissionaryLeft == 0 && X.numberCannibalLeft == 0 && X.boatSide == false)

{

return true;

}

return false;

}

}

}

namespace MissionaryProblem

{

public class State

{

public int numberMissionaryLeft;

public int numberCannibalLeft;

public int numberMissionaryRight;

public int numberCannibalRight;

public bool boatSide;

public State previousState;

public State(int ml, int cl, int mr, int cr, bool side, State previous)

{

this.numberMissionaryLeft = ml;

this.numberMissionaryRight = mr;

this.numberCannibalLeft = cl;

this.numberCannibalRight = cr;

this.boatSide = side;

this.previousState = previous;

}

public override string ToString()

{

return (boatSide ? "Left " : "Right") + " " + numberMissionaryLeft + "/" + numberCannibalLeft + " " + numberMissionaryRight + "/" + numberCannibalRight;

}

}

}

namespace MissionaryProblem

{

using System;

using System.Collections;

using System.Collections.Generic;

public class SolutionTest

{

public static void Main()

{

State start = new State(3, 3, 0, 0, true, null);

//List for all solution

State finish = BfsAlgorithm.BFS(start);

if (finish == null)

{

Console.WriteLine(("No solution was found."));

return;

}

//for (int i = 0; i < finish.Count; i++)

//{

//TODO: fix nextState = finish[i];

State nextState = finish;

Stack<State> path = new Stack<State>();

while (nextState != null)

{

path.Push(nextState);

nextState = nextState.previousState;

}

Console.WriteLine(("Solution path: "));

foreach (var item in path)

{

Console.WriteLine((item));

}

Console.WriteLine();

//}

}

}

}

5. Идея за корекция на псевдо-кода на алгоритъма, така, че да намира всички решения .

method BFS

създаване на опашка Open;

създаване на лист Closed;

създаване на лист AllSolution;

while (съдържанието на Open e различно от 0)

begin

Вадим най лявото състояние от Оpen и го поставяме в X;

if (X == Goal)

{

Довави в AllSolution Х;

if(съдържанието на Open e различно от 0)

{

Отиди на следваща итерация.

}

Else

{

Return AllSolution;

}

}

if (X се съдържа в Close)

{

Премини на следваща итерация

}

while(Генериране на наследници на Х)

определяме следващия наследник

if (проверяваме дали наследника отговаря на изискванията)

{

Добавя ме го в дясната страна на Open

}

If(ако не се съдържа в Open)

{

Поставяме Х в Closed;

}

return null;

}