Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

Організація обчислювальних процесів

Лабораторна робота №1

«Неінформований пошук»

Виконав:

студент групи ІС-73

Коноплянка Д. С.

Перевірила:

Мажара О. О.

Київ 2020 р.

**Мета:** Розробити алгоритм вирішення задачі про «Канібалів та місіонерів» із застосування алгоритму пошуку з ітеративним заглибленням.

Варіант №11 (1, 3)

1. Три місіонера і три канібала знаходяться на лівому березі річки. Всі хочуть перебратися на інший берег. Є невеликий човен, що вміщає не більше двох осіб. Якщо на якомусь березі канібалів виявиться більше, ніж місіонерів, то вони з’їдять місіонерів. Якщо виявиться більше місіонерів, то вони звернуть канібалів в свою віру. Знайти послідовність поїздок, що гарантує безпеку місіонерам і свободу віросповідання канібалів.

class Program

{

static void Main(string[] args)

{

Transition startTransition = new Transition(3, 3, Boat.Left, 0, 0, 0, 0);

Algorithm algorithm = new Algorithm();

Transition result = algorithm.IDDFS(startTransition);

if (result == null)

Console.WriteLine("No solution");

else

{

List<Transition> path = new List<Transition>();

Transition transition = result;

while (transition != null)

{

path.Add(transition);

transition = transition.parentSolution;

}

int depth = path.Count - 1;

int counter = 0;

for (int i = depth; i >= 0; i--)

{

transition = path[i];

if (transition.boat == Boat.LeftToRight || transition.boat == Boat.RightToLeft)

counter++;

if(transition.boat == Boat.Left || transition.boat == Boat.Right)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Cannibals L:{0}, Missioner L: {1}, --- Boat:{2} Cannibal:{3} Missioner: {4} --- " +

"Cannibal R: {5}, Missioner R: {6} ",

transition.cannibalLeft, transition.missionerLeft, transition.boat, transition.cannibalBoat, transition.missionerBoat, transition.cannibalRight, transition.missionerRight);

Console.ResetColor();

Console.WriteLine();

}

else

{

Console.WriteLine("Cannibals L:{0}, Missioner L: {1}, --- Boat:{2} Cannibal:{3} Missioner: {4} --- " +

"Cannibal R: {5}, Missioner R: {6} ",

transition.cannibalLeft, transition.missionerLeft, transition.boat, transition.cannibalBoat, transition.missionerBoat, transition.cannibalRight, transition.missionerRight);

}

}

Console.WriteLine("Green color - Near which shore is the boat");

Console.WriteLine();

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Depth: {0}", depth);

Console.WriteLine("Transitions: {0}", counter);

Console.WriteLine();

Console.ResetColor();

}

}

}

public enum Boat // Можливі стани човна

{

Right,

Left,

RightToLeft,

LeftToRight

}

public class Transition

{

public int cannibalLeft { get; set; }

public int cannibalRight { get; set; }

public int cannibalBoat { get; set; }

public int missionerLeft { get; set; }

public int missionerRight { get; set; }

public int missionerBoat { get; set; }

public Boat boat { get; set; }

public Transition parentSolution { get; set; }

public Transition(int cannibalLeft, int missionerLeft, Boat boat,

int cannibalRight, int missionerRight, int cannibalBoat, int missionerBoat)

{

this.cannibalLeft = cannibalLeft;

this.cannibalRight = cannibalRight;

this.cannibalBoat = cannibalBoat;

this.missionerLeft = missionerLeft;

this.missionerRight = missionerRight;

this.missionerBoat = missionerBoat;

this.boat = boat;

}

public bool Valid() // Умова для того щоб не було мінусових значень та щоб в човні не було білеше двух людей

{

if ((cannibalLeft >= 0 && cannibalRight >= 0 && missionerLeft >= 0 && missionerRight >= 0 && cannibalBoat >= 0 && missionerBoat >= 0) &&

(missionerRight == 0 || cannibalRight == missionerRight || cannibalRight == 0) &&

(missionerLeft == 0 || cannibalLeft == missionerLeft || cannibalLeft == 0) && (cannibalBoat + missionerBoat <= 2))

return true;

return false;

}

public bool Goal() // Рішення задачі

{

return cannibalLeft == 0 && missionerLeft == 0 && boat == Boat.Right;

}

public void CheckGoal(List<Transition> childTransition, Transition transition) // Перевірка валідності

{

if (transition.Valid())

{

transition.parentSolution = this;

childTransition.Add(transition);

}

}

public List<Transition> Transitions() // Можливі переходи

{

List<Transition> transitions = new List<Transition>();

if (boat == Boat.Left)

{

CheckGoal(transitions, new Transition(cannibalLeft - 2, missionerLeft, Boat.LeftToRight, cannibalRight, missionerRight, cannibalBoat + 2, missionerBoat));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft - 2, Boat.LeftToRight, cannibalRight, missionerRight, cannibalBoat, missionerBoat + 2));

CheckGoal(transitions, new Transition(cannibalLeft - 1, missionerLeft - 1, Boat.LeftToRight, cannibalRight, missionerRight, cannibalBoat + 1, missionerBoat + 1));

CheckGoal(transitions, new Transition(cannibalLeft - 1, missionerLeft, Boat.LeftToRight, cannibalRight, missionerRight, cannibalBoat + 1, missionerBoat));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft - 1, Boat.LeftToRight, cannibalRight, missionerRight, cannibalBoat, missionerBoat + 1));

}

if (boat == Boat.LeftToRight)

{

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.Right, cannibalRight + 2, missionerRight, cannibalBoat - 2, missionerBoat));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.Right, cannibalRight, missionerRight + 2, cannibalBoat, missionerBoat - 2));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.Right, cannibalRight + 1, missionerRight + 1, cannibalBoat - 1, missionerBoat - 1));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.Right, cannibalRight + 1, missionerRight, cannibalBoat - 1, missionerBoat));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.Right, cannibalRight, missionerRight + 1, cannibalBoat - 1, missionerBoat));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.RightToLeft, cannibalRight - 1, missionerRight + 2, cannibalBoat + 1, missionerBoat - 2));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.RightToLeft, cannibalRight + 2, missionerRight - 1, cannibalBoat - 2, missionerBoat + 1));

}

if (boat == Boat.Right)

{

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.RightToLeft, cannibalRight - 2, missionerRight, cannibalBoat + 2, missionerBoat));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.RightToLeft, cannibalRight, missionerRight - 2, cannibalBoat, missionerBoat + 2));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.RightToLeft, cannibalRight - 1, missionerRight - 1, cannibalBoat + 1, missionerBoat + 1));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.RightToLeft, cannibalRight - 1, missionerRight, cannibalBoat + 1, missionerBoat));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft, Boat.RightToLeft, cannibalRight, missionerRight - 1, cannibalBoat, missionerBoat + 1));

}

if (boat == Boat.RightToLeft)

{

CheckGoal(transitions, new Transition(cannibalLeft + 2, missionerLeft, Boat.Left, cannibalRight, missionerRight, cannibalBoat - 2, missionerBoat));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft + 2, Boat.Left, cannibalRight, missionerRight, cannibalBoat, missionerBoat - 2));

CheckGoal(transitions, new Transition(cannibalLeft + 1, missionerLeft + 1, Boat.Left, cannibalRight, missionerRight, cannibalBoat - 1, missionerBoat - 1));

CheckGoal(transitions, new Transition(cannibalLeft + 1, missionerLeft, Boat.Left, cannibalRight, missionerRight, cannibalBoat - 1, missionerBoat));

CheckGoal(transitions, new Transition(cannibalLeft, missionerLeft + 1, Boat.Left, cannibalRight, missionerRight, cannibalBoat, missionerBoat - 1));

CheckGoal(transitions, new Transition(cannibalLeft + 1, missionerLeft - 2, Boat.LeftToRight, cannibalRight, missionerRight, cannibalBoat - 1, missionerBoat + 2));

CheckGoal(transitions, new Transition(cannibalLeft - 2, missionerLeft + 1, Boat.LeftToRight, cannibalRight, missionerRight, cannibalBoat + 2, missionerBoat - 1));

}

return transitions;

}

}

public class Algorithm

{

public Transition DLS(Transition transition, int lim)

{

if (transition.Goal() && lim == 0)

return transition;

else

{

if( lim > 0)

{

List<Transition> transitions = transition.Transitions();

foreach (var item in transitions)

{

Transition result = DLS(item, lim - 1);

if (result != null)

return result;

}

}

}

return null;

}

public Transition IDDFS(Transition startTransition)

{

for (int i = 1; ; i++)

{

var result = DLS(startTransition, i);

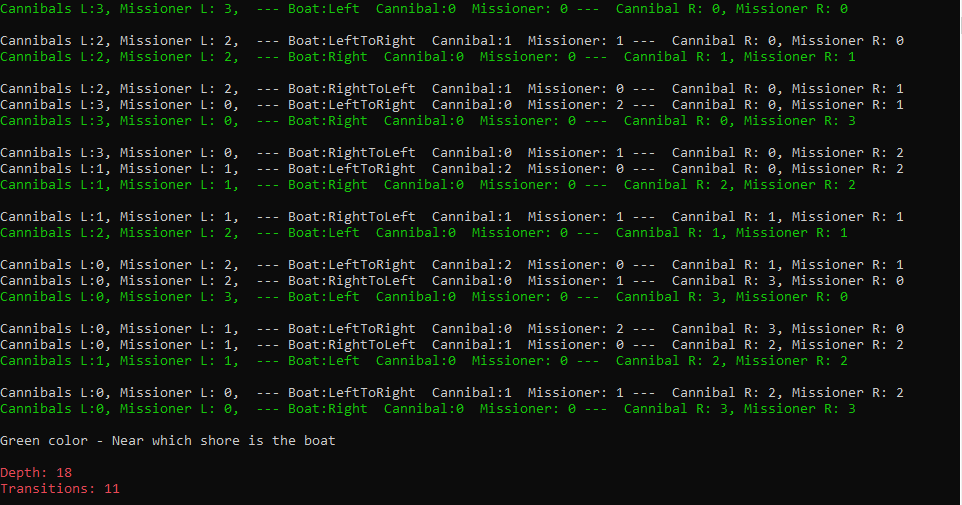
if (result != null)

return result;

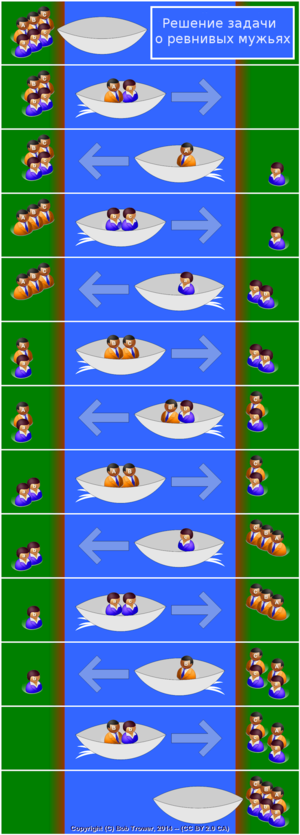
}

}

}



Висновок: В даній лабораторній роботі була вирішена задача про «Канібалів та Місіонерів» та визначена глубина яка становить 18. Число 11 це число переходів човна з одного берегу на інший

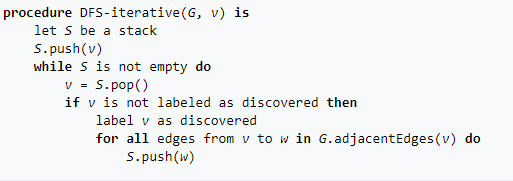


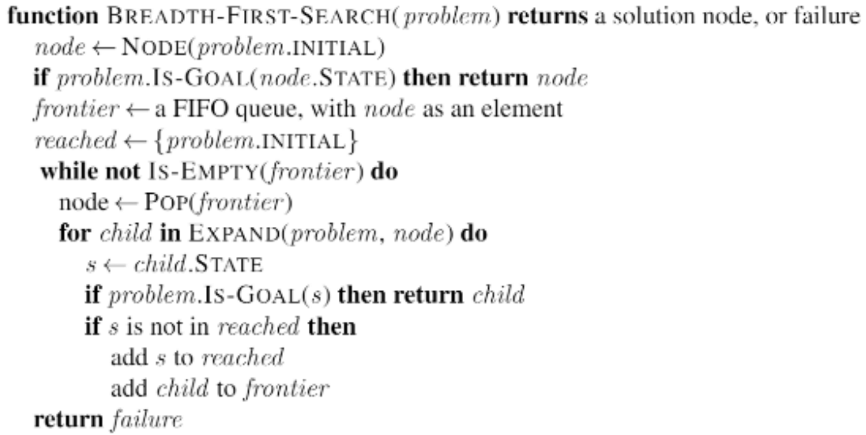
Контрольні запитання

1. Напишіть на псевдомові процедури пошуку в ширину і глибину, поясніть їх відмінність з алгоритмічної точки зору
2. Сформулюйте принципи пошуку, використовувані в алгоритмах пошуку в глибину:  з обмеженням глибини і з ітеративним поглибленням.
3. Які алгоритми неінформативного пошуку є повними? Оптимальними?

1) При реализации поиска в глубь можно дальше уйти от исходной вершины, а когда идти уже некуда, возвращаемся в ту вершину, откуда идет хотя бы одно ребро в не пройденные еще вершины.

При реализации поиска в ширь процесс примерно такой же только он идет в ширь, когда уже идти некуда переходим на следующий уровень





2) Поиск с ограничением глубины (DLS)

Если запустить поиск в глубину на бесконечном графе, то поиск никогда не завершиться. Для избежания этой ситуации можно добавить в стандартный DFS ограничение глубины - максимальную глубину на которую он может спускаться

**Поиск в глубину с итеративным углублением (IDDFS)**

Поиск в глубину редко применяется для поиска пути т.к. найденный путь не обязательно будет оптимальным. Но можно модифицировать поиск в глубину так, чтобы он искал оптимальный путь. Для этого циклично используется DLS, увеличивая ограничение глубины с каждой итерацией. В таком случае мы получим минимально необходимую глубину для нахождения пути.

3) Поиск в ширину( если действия имеют одинаковую стоимость), Поиск по критерию стоимости (если стоимость > 0), Поиск в глубину с итеративным углублением