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 №2

«Інформативний пошук»

Виконав:

студент групи ІС-73

Коноплянка Д. С.

Перевірила:

Мажара О. О.

Київ 2020 р.

**Мета:** Розробити алгоритм вирішення старовинної задачі про переливання води із застосування алгоритму A\*.

Варіант №11 (6, 1)

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

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

class Bucket

{

public int Capacity { set; get; }

public int Filled { set; get; }

}

class Node

{

public Node Prev { set; get; }

public Bucket FirstBucket { set; get; }

public Bucket SecondBucket { set; get; }

public string Action { set; get; }

public List<Node> GenerateNodes(Node node)

{

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

if (node.FirstBucket.Filled > 0)

{

if (node.SecondBucket.Filled != node.SecondBucket.Capacity)

nodes.Add(PourFirstToSecondBucket(node));

if (node.FirstBucket.Filled != node.FirstBucket.Capacity)

nodes.Add(FillFirstBucket(node));

nodes.Add(EmptyFirstBucket(node));

}

else

nodes.Add(FillFirstBucket(node));

if (node.SecondBucket.Filled > 0)

{

if (node.FirstBucket.Filled != node.FirstBucket.Capacity)

nodes.Add(PourSecondToFirstBucket(node));

if (node.SecondBucket.Capacity != node.SecondBucket.Filled)

nodes.Add(FillSecondBucket(node));

nodes.Add(EmptySecondBucket(node));

}

else

nodes.Add(FillSecondBucket(node));

return nodes;

}

public Node Algoritm(Node node)

{

List<Node> nodes = GenerateNodes(node);

while (true)

{

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

foreach (var item in nodes)

{

temp.AddRange(GenerateNodes(item));

}

temp.RemoveAll(x => (x.FirstBucket.Filled == 9 && x.SecondBucket.Filled == 5) ||

(x.FirstBucket.Filled == 0 && x.SecondBucket.Filled == 0));

nodes = temp;

foreach (var item in nodes)

{

if (IsGoal(item))

{

return item;

}

}

}

}

static bool IsGoal(Node node)

{

var result = 3;

return node.FirstBucket.Filled == result || node.SecondBucket.Filled == result;

}

public Node FillFirstBucket(Node node)

{

return new Node()

{

Prev = node,

FirstBucket = new Bucket() { Capacity = 9, Filled = 9 },

SecondBucket = new Bucket() { Capacity = node.SecondBucket.Capacity, Filled = node.SecondBucket.Filled },

Action = "Fill the first Bucket "

};

}

public Node FillSecondBucket(Node node)

{

return new Node()

{

Prev = node,

FirstBucket = new Bucket() { Capacity = node.FirstBucket.Capacity, Filled = node.FirstBucket.Filled },

SecondBucket = new Bucket() { Capacity = 5, Filled = 5 },

Action = "Fill the second Bucket"

};

}

public Node EmptyFirstBucket(Node node)

{

return new Node()

{

Prev = node,

FirstBucket = new Bucket() { Capacity = node.FirstBucket.Capacity, Filled = 0 },

SecondBucket = new Bucket() { Capacity = node.SecondBucket.Capacity, Filled = node.SecondBucket.Filled },

Action = "Empty the first bucket"

};

}

public Node EmptySecondBucket(Node node)

{

return new Node()

{

Prev = node,

FirstBucket = new Bucket() { Capacity = node.FirstBucket.Capacity, Filled = node.FirstBucket.Filled },

SecondBucket = new Bucket() { Capacity = node.SecondBucket.Capacity, Filled = 0 },

Action = "Empty the second bucket "

};

}

public Node PourFirstToSecondBucket(Node node)

{

var firstBucket = new Bucket() { Capacity = node.FirstBucket.Capacity };

var secondBucket = new Bucket() { Capacity = node.SecondBucket.Capacity };

var freeSpaceInSecond = node.SecondBucket.Capacity - node.SecondBucket.Filled;

if (node.FirstBucket.Filled >= freeSpaceInSecond)

{

secondBucket.Filled = 5;

firstBucket.Filled = node.FirstBucket.Filled - freeSpaceInSecond;

}

else

{

secondBucket.Filled += node.FirstBucket.Filled;

firstBucket.Filled = 0;

}

return new Node()

{

Prev = node,

FirstBucket = firstBucket,

SecondBucket = secondBucket,

Action = "Pour First To Second Bucket"

};

}

public Node PourSecondToFirstBucket(Node node)

{

var firstBucket = new Bucket() { Capacity = node.FirstBucket.Capacity };

var secondBucket = new Bucket() { Capacity = node.SecondBucket.Capacity };

var freeSpaceInFirst = node.FirstBucket.Capacity - node.FirstBucket.Filled;

if (node.SecondBucket.Filled >= freeSpaceInFirst)

{

firstBucket.Filled = 9;

secondBucket.Filled = node.SecondBucket.Filled - freeSpaceInFirst;

}

else

{

firstBucket.Filled += node.SecondBucket.Filled;

secondBucket.Filled = 0;

}

return new Node()

{

Prev = node,

FirstBucket = firstBucket,

SecondBucket = secondBucket,

Action = "Pour Second To First Bucket"

};

}

}

class Program

{

static void Main(string[] args)

{

var firstBucket = new Bucket()

{

Capacity = 9,

Filled = 0

};

var secondBucket = new Bucket()

{

Capacity = 5,

Filled = 0

};

var node = new Node()

{

FirstBucket = firstBucket,

SecondBucket = secondBucket

};

var sol = node.Algoritm(node);

if (sol != null)

{

Print(sol);

}

}

static void Print(Node node)

{

if (node.Prev != null)

{

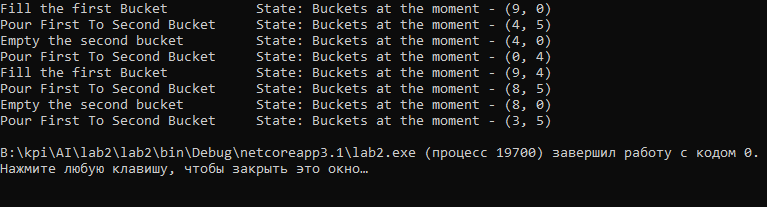
Print(node.Prev);

Console.WriteLine(node.Action + $"\tState: Buckets at the moment - ({node.FirstBucket.Filled}, {node.SecondBucket.Filled})");

}

}

}

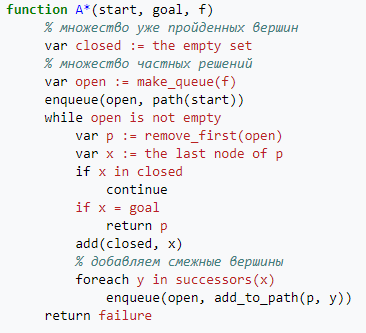


Висновок: В даній лабораторній роботі була вирішена задача про «переливання води». Алгоритм надав один із найкоротших шляхів

Контрольні запитання

1. Напишіть на псевдомові процедури пошуку A\*, RBFS
2. Яка відмінність між алгоритмами інформативного та неінформативного пошуку?
3. Поясніть необхідність використання евристичних функцій при інформативному пошуку.
4. 4. Якою повинна бути евристична функція для того, щоб алгоритми інформативного пошуку були оптимальними?
5. Які алгоритми інформативного пошуку є повними? Оптимальними? Чому?
6. 6. Поясніть принцип роботи алгоритмів, які реалізовані у лабораторній роботі.

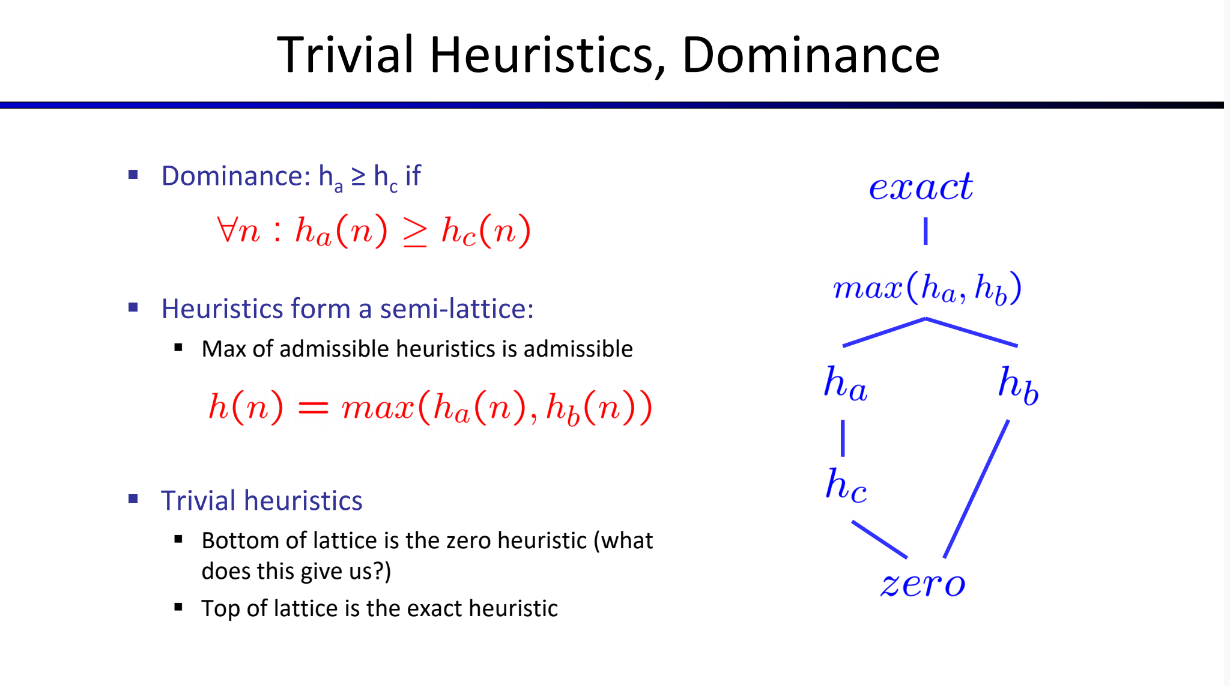
1)

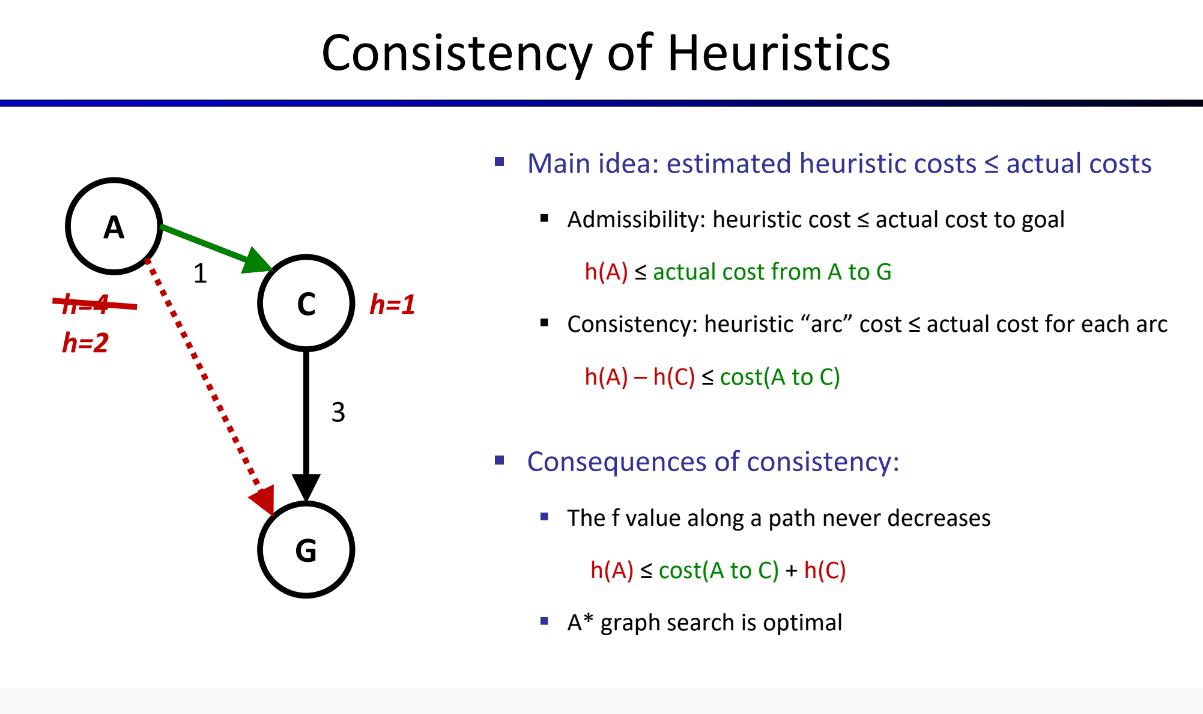


2) Інформативні методи зазвичай забезпечують більш ефективний пошук по порівнянні з неінформованими методами тому що, застосовується стратегія пошуку рішень в просторі станів, в якій використовуються знання, що відносяться до конкретного завдання

3) Дозволяє прискорити вирішення завдання в тих випадках, коли точне рішення не може бути знайдено

4) Евристична функція на кожному кроці перебору оцінює альтернативи на підставі додаткової інформації з метою прийняття рішення про те, в якому напрямку слід продовжувати перебір





5) Поиск A\*  
У ньому застосовується оцінка f (n) вартості найменш дорогого шляхи вирішення, що проходить через вузол n:

f (n) = g (n) + h (n), де

g (n) - вартість шляху від початкового вузла до вузла n,

h (n) - оцінка вартості шляху від вузла n до мети.

Якщо h (n) ніколи не переоцінює вартість досягнення мети (тобто є допустимою), то пошук A \* є оптимальним.

6) Порядок обходу вершин визначається евристичної функцією «відстань + вартість» (зазвичай позначається як f (x)). Ця функція - сума двох інших: функції вартості досягнення даної вершини (x) з початковою (зазвичай позначається як g (x) і може бути як евристичної, так і немає), і функції евристичної оцінки відстані від розглянутої вершини до кінцевої (позначається як h (x)).

Функція h (x) повинна бути допустимої евристичної оцінкою, тобто не повинна переоцінювати відстані до цільової вершині. Наприклад, для завдання маршрутизації h (x) може являти собою відстань до цілі по прямій лінії, так як це фізично найменша можлива відстань між двома точками.