Прізвище: Долінський
Ім'я: Олег
Група: КН-406
Варіант: 8
Кафедра: САПР
Дисципліна: Дискретні моделі в САПР
Прийняв: Кривий Р.З.
Посилання: https://github.com/olehdol/labs.git

ЗВІТ

до лабораторної роботи №3

на тему «**Задача комівояжера, метод гілок і границь**»

Мета роботи

Отримати практичні навички роботи з пошуку рішення задачі комівояжера.

Короткі теоретичні відомості

Основні правила алгоритму можуть бути сформульовані наступним чином:

1. розгалуження в першу чергу піддається підмножина з номером, якому відповідає найменше значення нижньої оцінки цільової функції (I - це безліч номерів всіх підмножин, (або), що знаходяться на кінцях гілок і розгалуження яких ще не припинено). Якщо реалізується викладений вище спосіб розгалуження множин, то може виникнути неоднозначність щодо вибору компоненти, по якій необхідно здійснювати черговий крок розгалуження. На жаль, питання про «найкращому» способі такого вибору з загальних позицій поки не вирішене, і тому в конкретних завданнях використовуються деякі евристичні правила.

2. Якщо для деякого і-го підмножини виконується умова, то розгалуження його необхідно припинити, так як потенційні можливості знаходження гарного рішення в цьому підмножині (їх характеризує) виявляються гірше, ніж значення цільової функції для реального, знайденого до даного моменту часу, допустимого рішення вихідної завдання (воно характеризує).

3. Галуження підмножини припиняється, якщо знайдене в завданню (4) оптимальне рішення. Обґрунтовується це тим, що, і, отже, кращого допустимого рішення, ніж в цьому підмножині не існує. В цьому випадку розглядається можливість коригування.

4. Якщо, де, то виконуються умови оптимальності для знайденого до цього моменту кращого допустимого рішення. Обгрунтування таке ж, як і пункту 2 цих правил.

5. Після знаходження хоча б одного допустимого рішення вихідної завдання може бути розглянута можливість зупинки роботи алгоритму з оцінкою близькості кращого з отриманих допустимих рішень до оптимального (за значенням цільової функції):

Індивідуальне завдання:

```
6
0 0 69 60 10 20
0 0 0 31 39 2
69 0 0 0 59 0
60 31 0 0 0 36
10 39 59 0 0 79
20 2 0 36 79 0
```

Виконання:

Приведення матриці витрат і обчислення нижньої оцінки вартості маршруту r.

1. Обчислюємо найменший елемент в кожному рядку (константа приведення для рядка)

2. Переходимо до нової матриці витрат, віднімаючи з кожного рядка її константу приведення

3. Обчислюємо найменший елемент в кожному стовпці (константа приведення для стовпця)

4. Переходимо до нової матриці витрат, віднімаючи з кожного стовпчика його константу приведення.

Як результат маємо матрицю витрат, в якій в кожному рядку і в кожному стовпці є хоча б один нульовий елемент.

5. Обчислюємо кордон на даному етапі як суму констант приведення для стовпців і рядків (дана межа буде вартістю, менше якої неможливо побудувати шуканий маршрут)

Знайти мінімальні значення у стовпці та рядку:

```
private static int FindMinWeightInRow(int[,] matrix, int row)
{
    var min = int.MaxValue;
    for (int i = 0; i < matrix.GetLength(1); i++)
    {
        if (matrix[row, i] != -1 && matrix[row, i] < min)
            min = matrix[row, i];
    }
    return min;
}
2 references
private static int FindMinWeightInCol(int[,] matrix, int col)
{
    var min = int.MaxValue;
    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        if (matrix[i, col] != -1 && matrix[i, col] < min)
            min = matrix[i, col];
    }
    return min;
}
```

Видалити мінімальні значення у стовпці та рядку:

```
private static int[,] SubstractFromRow(int[,] matrix, int row, int minForRow)
{
    for (int i = 0; i < matrix.GetLength(1); i++)
    {
        if (matrix[row, i] != -1)
            matrix[row, i] -= minForRow;
    }
    return matrix;
}
1 reference
private static int[,] SubstractFromCol(int[,] matrix, int col, int minForCol)
{
    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        if (matrix[i, col] != -1)
            matrix[i, col] -= minForCol;
    }
    return matrix;
}
```

Пошук мінімальної ваги в кожному рядку та стовці. Видалення цього мінімуму із кожного рядка і стовпця.

```
private static int[,] Substracting(int[,] matrix)
{
    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        var minForRow = FindMinWeightInRow(matrix, i);
        if (minForRow != int.MaxValue)
        {
            matrix = SubstractFromRow(matrix, i, minForRow);
            _cost += minForRow;
            noValues = false;
        }
    }
    for (int i = 0; i < matrix.GetLength(1); i++)
    {
        var minForCol = FindMinWeightInCol(matrix, i);
        if (minForCol != int.MaxValue)
        {
            matrix = SubstractFromCol(matrix, i, minForCol);
            _cost += minForCol;
            noValues = false;
        }
    }
    return matrix;
}
```

1.Обчислення штрафу за невикористання для кожного нульового елемента наведеної матриці витрат.

- Шукаю усі нульові елементи в наведеній матриці
- Для кожного з них вважаємо його штраф за невикористання.

- Вибираю елемент, якому відповідав би максимальний штраф (будь-який, якщо їх декілька)

```
for (int i = 0; i < matrix.GetLength(0); i++)
{
    for (int j = 0; j < matrix.GetLength(1); j++)
    {
        int coef = default;
        if (matrix[i, j] == 0)
        {
            coef = CalculateCoeficient(matrix, i, j);
            if (coef > maxCoef)
            {
                maxCoefForRow = i;
                maxCoefForCol = j;
                maxCoef = coef;
                if (Fine.Exists(x => x.row == maxCoefForRow && x.col == maxCoefForCol))
                {
                    var tuple = Fine.First(x => x.row == maxCoefForRow && x.col == maxCoefForCol);
                    tuple.maxCof = coef;
                    var index = Fine.FindIndex(x => x.row == maxCoefForRow && x.col == maxCoefForCol);
                    Fine[index] = tuple;
                }
                else
                    Fine.Add((i, j, coef));
            }
        }
    }
}
```

4. З усіх нерозбитих множин вибирається те, яке має найменшу оцінку.

```
if (costWithoutEdge < costWithEdge)
{
    var rollbackMatrix = matrices.First(x => x.pathCost < costWithEdge).matrix;
    int a = 0; int b = 0; int m = 0;
    rollbackMatrix = RemoveMaxCoef(rollbackMatrix, out a, out b, out m);
    rollbackMatrix[a, b] = -1;
    BranchAndBound(rollbackMatrix);
}
else
{
    _edges.Add(new Edge()
    {
        Source = r,
        Destination = c,
        Weight = StaticMatrix[r, c]
    });
    if (_edges.Count >= 2 && _edges.SkipLast(1)
        .Last().Source == _edges.Last().Destination)
        matrix[_edges.SkipLast(1)
            .Last().Destination, _edges.Last().Source] = -1;
    lowestCost = costWithEdge;
    BranchAndBound(matrix);
}
```

Результати:

Завдання задане індивідуальним варіантом не вийшло розв’язати, оскільки в шляху появляються грані, котрих немає у початкові матриці, це можна помітити у результаті виконання.

Тому, було відредаговано(замінені нульові ваги) початкову матрицю, відредагована матриця мала розв'язок. Матриця та результат розв'язання зображений нижче:

```
Задача листоноші
Початкова матриця
        -1      33      69      60      10      20
        33      -1      44      31      39      2
        69      44      -1      55      59      22
        60      31      55      -1      66      36
        10      39      59      66      -1      79
        20      2       22      36      79      -1

Шлях

0 ==> 4 ==> 5 ==> 0 ==> 2 ==> 5 ==> 1 ==> 3 ==> 3 ==> 2 ==> 4 ==> 1
```

Висновок:

Розв’язано задачу комівояжера за допомогою методу гілок та меж. Було знайдено гамільтонів цикл, проте, цей цикл включав неіснуючі ребра. В початковій матриці була задана матриця із пропущеними ребрами у графі(неіснуючими), було відредаговано матрицю таким чином, щоб існували усі ребра, окрім петель. Результат розв’язання зображено на рисунку.