



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт компьютерных наук
Кафедра автоматизированных систем управления

Лабораторная работа №2
по системному анализу

Студент АС-21-1 _____ Станиславчук С. М.
(подпись, дата)

Руководитель
Профессор, к.т.н. _____ Качановский Ю. П.
(подпись, дата)

Липецк 2024

Содержание

1. Задание кафедры, соответствующее варианту, номер варианта
2. Алгоритм работы программы
3. Тестовый пример (в виде скриншота)

1. Задание кафедры, соответствующее варианту, номер варианта

Вариант 2.

Задание. Программирование алгоритма выделения иерархических уровней в структурной модели системы (введения порядковой функции на графе без контуров). Использовать исходное описание графа. После изменения нумерации вершин указать новый и старый номер вершины и представить новый граф в новом заданном описании

Задано: матрица смежности (A)

Получить: Множество левых инцидентов G^-

2. Алгоритм работы программы

Нам дано задание, где нужно упорядочить граф по входной матрице смежности, показать преобразование старого -> нового узла, а также получить из новой отсортированной матрицы множество левых инцидентов.

Выглядит, конечно, сложно, но если разбить задание на подзадачи все станет намного яснее:

1. Для начала разработаем алгоритм выделения иерархии уровней графа (выделим его в методе CreateHLevel(), в который будем передавать ребра графа, число вершин(необязательно, можно получить и из списка ребер), а на выходе получим двумерный список, в котором будут храниться наши уровни)

```
public static void CreateHLevel(List<HierarchicalLevel> HL, int numberV, List<Edge> E)
//numberV - количество вершин
{
    List<int> usedV = new(); //список вершин, уже использованных в порядковой функции

    List<int> notUsedV = new(); //список вершин, еще не использованных в порядковой функции

    for (int i = 0; i < numberV; i++)
        notUsedV.Add(i);

    while (notUsedV.Count > 0)
    {
        HL.Add(new HierarchicalLevel());

        for (int i = 0; i < notUsedV.Count; i++)
        {
```

```

        int k = 0;

        for (int j = 0; j < E.Count; j++)
            if (E[j].v2 == notUsedV[i])
                k++; //считаем полустепень захода вершины

        for (int m = 0; m < usedV.Count; m++)
            for (int j = 0; j < E.Count; j++)
            {
                if (E[j].v1 == usedV[m] && E[j].v2 == notUsedV[i])
                    k--; //вычитаем дуги, исходящие из вершин предыдущих уровней и
    входящие в вершину i
            }

        if (k == 0)
        {
            HL[HL.Count - 1].Level.Add(notUsedV[i]);
            notUsedV.RemoveAt(i);
            i--;
        }
    }

    for (int j = 0; j < HL[HL.Count - 1].Level.Count; j++)
    {
        usedV.Add(HL[HL.Count - 1].Level[j]);
    }
}
}

```

2. Полученные иерархические уровни нужно отсортировать в возрастающем порядке

```

List<HierarchicalLevel> normalizedSortedLevels = new();

CreateHLevel(normalizedSortedLevels, lines.Length, edges);

int ordereredVert = 1;

for (int i = 0; i < levels.Count; i++)
{
    for (int j = 0; j < normalizedSortedLevels[i].Level.Count; j++)
    {
        normalizedSortedLevels[i].Level[j] = ordereredVert++;
    }
}

```

3. С помощью отсортированных иерархических уровней, а также замены старых ребер на новые, получим матрицу смежности

```
private List<Edge> GenerateSortedEdges(List<Edge> oldEdges, List<VertsTransformation>
vertsTransformations)

{
    List<Edge> newEdges = new();

    // Проходимся по всем старым ребрам
    foreach (Edge oldEdge in oldEdges)
    {
        int oldV1 = oldEdge.v1;
        int oldV2 = oldEdge.v2;

        // Находим новый узел для первой вершины ребра
        int newV1 = FindNewVertex(oldV1+1, vertsTransformations);

        // Находим новый узел для второй вершины ребра
        int newV2 = FindNewVertex(oldV2+1, vertsTransformations);

        // Создаем новое ребро и добавляем его в список новых ребер
        newEdges.Add(new Edge(newV1, newV2));
    }

    return newEdges;
}

// Метод для нахождения нового узла на основе старого и списка vertsTransformations
private int FindNewVertex(int oldVertex, List<VertsTransformation> vertsTransformations)
{
    // Проходимся по всем уровням
    foreach (VertsTransformation oldNewVerts in vertsTransformations)
    {
        if (oldNewVerts.Old == oldVertex)
        {
            // Debug.Log($"return oldNewVerts.New = {oldNewVerts.New-1}");

            // Возвращаем новый узел
            return oldNewVerts.New-1;
        }
    }

    Debug.Log($"return oldVertex = {oldVertex}");
}
```

```

        // Если новый узел не найден, возвращаем старый
        return oldVertex;
    }

    // Метод для генерации матрицы смежности на основе списка ребер
    public static int[,] GenerateAdjacencyMatrix(List<Edge> edges)
    {
        int maxVertex = 0;

        foreach (Edge edge in edges)
        {
            maxVertex = Mathf.Max(maxVertex, Mathf.Max(edge.v1+1, edge.v2+1));
        }

        int[,] adjacencyMatrix = new int[maxVertex, maxVertex];

        foreach (Edge edge in edges)
        {
            adjacencyMatrix[edge.v1, edge.v2] = 1;
        }

        return adjacencyMatrix;
    }

```

4. Остается только вычислить множество левых инцидентов для полученной матрицы смежности

```

private static Dictionary<int, List<int>> LeftIncidence(int[,] adjacencyMatrix)
{
    int numNodes = adjacencyMatrix.GetLength(0);

    Dictionary<int, List<int>> NEmap = new(numNodes);

    for (int i = 0; i < numNodes; i++)
    {
        List<int> edges = new();
        for (int j = 0; j < numNodes; j++)
        {
            if (adjacencyMatrix[j, i] == 1)
            {
                edges.Add(j + 1);
            }
        }
    }
}

```

```

        int node = i + 1;

        NEmap.Add(node, edges);
    }

    return NEmap;
}

```

3. Тестовый пример

Error text will be here...

Hey!

Generate

0	1	0	0	0	0	0
0	0	0	0	0	1	0
1	0	0	0	1	0	1
0	0	0	0	0	1	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0

Sorted matrix

0	1	1	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	0	1
0	0	0	0	0	0	0

new 1(old 3)
new 2(old 1)
new 3(old 5)

Result

Left incidences (G_i)

- $G(1) = \{\}$
- $G(2) = \{1\}$
- $G(3) = \{1\}$
- $G(4) = \{1\}$
- $G(5) = \{2; 3\}$
- $G(6) = \{4\}$
- $G(7) = \{5; 6\}$

Error text will be here...

Hey!

Generate

0	1	0	0	0	0	0
0	0	0	0	0	1	0
1	0	0	0	1	0	1
0	0	0	0	0	1	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0

Sorted matrix

0	0	0	0	0	0	1
0	0	0	0	0	0	0

new 1(old 3)
new 2(old 1)
new 3(old 5)
new 4(old 7)
new 5(old 2)
new 6(old 4)
new 7(old 6)

Result

Left incidences (G_i)

- $G(1) = \{\}$
- $G(2) = \{1\}$
- $G(3) = \{1\}$
- $G(4) = \{1\}$
- $G(5) = \{2; 3\}$
- $G(6) = \{4\}$
- $G(7) = \{5; 6\}$

В колонке “Sorted matrix” появилась отсортированная матрица смежности, а ниже список новых и старых вершин.

Вывод: написал ПО, способное сортировать граф, описанный матрицей смежности, получать на его основе множества левых инцидентов.