Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА – Российский технологический университет"
РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

Кафедра КБ-14 «Цифровые технологии обработки данных»

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №2

«Реализация алгоритма на графах»

по дисциплине

«Алгоритмы и структуры данных»

Вариант № 6

Выполнил: студент 2 курса
группы БСБО-11-21
шифр 21Б0296
Хохлов Дмитрий Владимирович

Проверил:
Головин Леонид Олегович

Москва 2022 г.

Задание на лабораторную работу № 2.

В рамках лабораторной работы №1 требуется реализовать в виде программы абстрактный тип данных «Граф» согласно варианту с учетом заданного представления графа. Операторы (операции) АТД «Граф» функционально должны выполнять следующие операции (названия операций – примерные):

1. FIRST(v) - возвращает индекс первой вершины, смежной с вершиной v. Если вершина v не имеет смежных вершин, то возвращается "нулевая" вершина □.

2. NEXT(v, i)- возвращает индекс вершины, смежной с вершиной v, следующий за индексом i.

Если i — это индекс последней вершины, смежной с вершиной v, то возвращается □.

3. VERTEX(v, i) - возвращает вершину с индексом i из множества вершин, смежных с v.

4. ADD_V(<имя>,<метка, mark>) - добавить УЗЕЛ

5. ADD_E(v, w, c) - добавить ДУГУ (здесь c — вес, цена дуги (v,w))

6. DEL_V(<имя>) - удалить УЗЕЛ

7. DEL_E(v, w) – удалить ДУГУ

8. EDIT_V(<имя>, <новое значение метки или маркировки>) - изменить метку (маркировку)

УЗЛА

EDIT_E(v, w, <новый вес дуги>) - изменить вес ДУГИ

Вариант № 96.

Алгоритм: Методом обхода в глубину вычислить цикломатическую сложность графа

Способ представления графа: Матрица инцидентности

Термины

Что такое обход графа?

Простыми словами, обход графа — это переход от одной его вершины к другой в поисках свойств связей этих вершин. Связи (линии, соединяющие вершины) называются направлениями, путями, гранями или ребрами графа. Вершины графа также именуются узлами.

Матрица инцидентности - одна из форм представления графа, в которой указываются связи между инцидентными элементами графа (ребро(дуга) и вершина). Столбцы матрицы соответствуют ребрам, строки — вершинам. Ненулевое значение в ячейке матрицы указывает связь между вершиной и ребром (их инцидентность).

Цикломатическая сложность части программного кода — количество линейно независимых маршрутов через программный код. Например, если исходный код не содержит никаких точек ветвления или циклов, то сложность равна единице, поскольку есть только единственный маршрут через код. Если код имеет единственный оператор IF, содержащий простое условие, то существует два пути через код: один если условие оператора IF имеет значение TRUE и один — если FALSE

Математически цикломатическая сложность структурированной программы определяется с помощью ориентированного графа, узлами которого являются блоки программы, соединенные рёбрами, если управление может переходить с одного блока на другой. Тогда сложность определяется как:

$M = E - N + 2P$,

где:

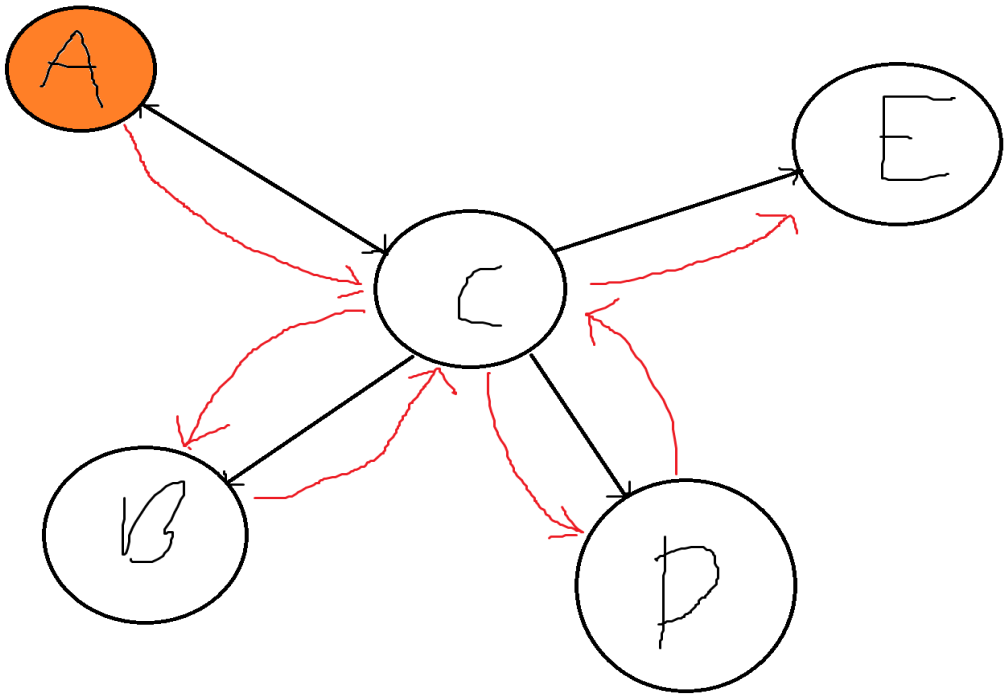
M = цикломатическая сложность,

E = количество рёбер в графе,

N = количество узлов в графе,

P = количество компонент связности.

Пример графа с выполнением алгоритма.



Начинать будем с вершины A
В данном случае обход в глубину произойдет по пути ACBDE.

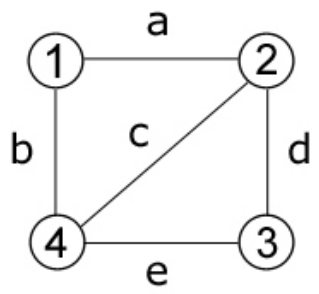
Граф будет иметь 1 компонент связи - вершину E, вершин у на 5, а путей 4, следовательно по формуле M = E - N + 2P, получаем M = 5 - 4 + 2*1 = 3

С какой бы вершины мы не начинали результат будет то же самый

Способ представления графа.

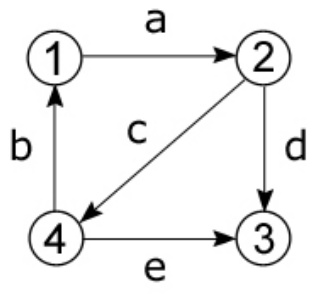
Матрица инцидентности строиться по похожему, но не по тому же принципу, что и матрица смежности. Так если последняя имеет размер n×n, где n – число вершин, то матрица инцидентности – n×m, здесь n – число вершин графа, m – число ребер. То есть чтобы задать значение какой-либо ячейки, нужно сопоставить не вершину с вершиной, а вершину с ребром. В каждой ячейки матрицы инцидентности неориентированного графа стоит 0 или 1, а в случае ориентированного графа, вносятся 1, 0 или -1.

	a	b	c	d	e
1	1	1	0	0	0
2	1	0	1	1	0
3	0	0	0	1	1
4	0	1	1	0	1



Матрица инцидентности | Рисунок 1 | Неориентированный граф

	a	b	c	d	e
1	1	-1	0	0	0
2	-1	0	1	1	0
3	0	0	0	-1	-1
4	0	1	-1	0	1



Матрица инцидентности | Рисунок 2 | Ориентированный граф

Сравнивая ориентированный и неориентированный графы, можно заметить, что некоторые положительные единицы сменились на отрицательные, например, в неориентированном графе ячейка (1, b) содержит 1, а в орграфе -1. Это уместно, т. к. в первом случае ребро b не направленное, а во втором – направленное.

Каждый столбец отвечает за какое-либо одно ребро, поэтому граф, описанный при помощи матрицы инцидентности, всегда будет иметь следующий признак: любой из столбцов матрицы инцидентности содержит две единицы, либо 1 и -1 когда это ориентированное ребро, все остальное в нем – нули.

Листинг программы.

```
let readline= require('readline');



vertexes = ["A","B","C","D","E"]



edges = [
      /*A,B,C,D,E*/
   /*A*/[0,0,1,0,0],
   /*B*/[0,0,1,0,0],
   /*C*/[1,1,0,1,1],
   /*D*/[0,0,1,0,0],
   /*E*/[0,0,1,0,0],
]



let vertexCount = ''
let P = 0
let P_Array = []



const DFS = (j, count) => {
   vertexCount+=vertexes[j]
   count = 0
   for (let i = 0; i < edges[j].length; i++) {
      if(i == j) continue



      if(edges[j][i] == 1 && !vertexCount.includes(vertexes[i])){
         count++
```

```
            DFS(i,count)
        }
    }
    P_Array[P_Array.length] = count
}


const lisen = () => {


    let rl = readline.createInterface({
        input: process.stdin,
        output: process.stdout,
        prompt: '>>>'
    });
    rl.prompt();


    rl.on('line', (input) => {


        if(input.includes("FIRST")){


            c1 = input.indexOf("(")
            c2 = input.indexOf(")")


            v_i = vertexes.indexOf(input.slice(c1+1,c2).trim())


            findVertex_i = edges[v_i].join('').indexOf(1)
            findVertex_i = findVertex_i == -1 ? 0 : findVertex_i
```

```
    console.log(vertexes[findVertex_i], findVertex_i)
}


if(input.includes("NEXT")){


    c1 = input.indexOf("(")
    c2 = input.indexOf(")")

    let string = ''

    for (let i = c1+1; i < c2; i++)
        if(input[i] != " ")
            string +=input[i]

    let [vertex, index] = string.split(',')

    v_i = vertexes.indexOf(vertex)
    ways = edges[v_i].join('').substring(+index+1).indexOf(1)

    findVertex_i = ways == -1 ? 0 : ways + +index + 1
    findVertex_i = findVertex_i == -1 ? 0 :findVertex_i

    console.log(vertexes[findVertex_i], findVertex_i)
}
if(input.includes("VERTEX")){
    c1 = input.indexOf("(")
    c2 = input.indexOf(")")
```

```
    let string = ''
    let count = 0
    let find

    for (let i = c1+1; i < c2; i++)
        if(input[i] != " ")
            string +=input[i]

    let [vertex, index] = string.split(',')
    let v_i = vertexes.indexOf(vertex)

    for (let i = 0;i < vertexes.length; i++)
        if(edges[v_i][i] !=0 && count <= index-1){
            count++
            find = i
        }
    console.log(edges[v_i][find], vertexes[find])
}

if(input.includes("ADD_V")){
    c1 = input.indexOf("(")
    c2 = input.indexOf(")")||input[input.length-1]

    vertexes[vertexes.length] = input.slice(c1+1,c2).trim()

    edges.map((edge)=> edge[edge.length] = 0)
    edges[edges.length] = []

    for (let i = 0; i < vertexes.length; i++)
        edges[edges.length-1][i] = 0
```

```
}
if(input.includes("ADD_E")){

    c1 = input.indexOf("(")
    c2 = input.indexOf(")")

    let vertex = ''

    for (let i = c1+1; i < c2; i++)
        if(input[i] != " ")
            vertex+=input[i]

    let [editStart, editEnd, way] = vertex.split(',')

    editStart_i = vertexes.join('').indexOf(editStart)
    editEnd_i   = vertexes.join('').indexOf(editEnd)

    edges[editStart_i][editEnd_i] = Number(way)
}
if(input.includes("DEL_V")){
    c1 = input.indexOf("(")
    c2 = input.indexOf(")")

    vertex_i = vertexes.join('').indexOf(input.slice(c1+1,c2).trim())
    vertexes.splice(vertex_i, 1);

    edges.map( edge => edge.splice(vertex_i, 1))
    edges.splice(vertext_i, 1)
}
if(input.includes("DEL_E")){
```

```
    c1 = input.indexOf("(")
    c2 = input.indexOf(")")

    let vertex = ''

    for (let i = c1+1; i < c2; i++)
        if(input[i] != " ")
            vertex+=input[i]

    let [editStart, editEnd] = vertex.split(',')

    editStart_i = vertexes.join('').indexOf(editStart)
    editEnd_i   = vertexes.join('').indexOf(editEnd)

    edges[editStart_i][editEnd_i] = 0
}
if(input.includes("EDIT_V")){
    c1 = input.indexOf("(")
    c2 = input.indexOf(")")

    vertexes[vertexes.length] = input.slice(c1+1,c2).trim()

    edges.map((edge)=> edge[edge.length] = 0)

    for (let i = 0; i < vertexes.length; i++){
                edges[edges.length-1][i] = 0
        }
}
if(input.includes("EDIT_E")){
    c1 = input.indexOf("(")
    c2 = input.indexOf(")")
```

```
    let vertex = ''

    for (let i = c1+1; i < c2; i++) {
                if(input[i] != " ") {
                        vertex+=input[i]
      }

      }

    let [editStart, editEnd, way] = vertex.split(',')

    editStart_i = vertexes.join('').indexOf(editStart)
    editEnd_i   = vertexes.join('').indexOf(editEnd)

    edges[editStart_i][editEnd_i] = Number(way)
}
if(input.includes("SH_V")) console.log(vertexes)
if(input.includes("SH_E")) console.log(edges)

if(input.includes("CYCL_COM")) {

    vertexCount = ''
    P=0
    P_Array = []

    DFS(2,0)

    let indexes = []
```

```
            edges.forEach(ary => {
                indexes = []

                let j = -1

                while((j = ary.indexOf(1, j + 1)) !== -1) {
indexes.push(j)
                }
            })

            P+= indexes.length == 0? 1 : 0

            console.log(vertexCount)

            console.log(`Цикломатическую сложность: ${vertexCount.length - (vertexCount.length-1) +2*P }`)
        }
        rl.close();

        if(!input.includes("STOP"))
            lisen()
    });
}
lisen()
```

O(f(n)) = n

Пример выполнения программы.

```
>>>SH_V
[ 'A', 'B', 'C', 'D', 'E' ]
>>>SH_E
[
  [ 0, 0, 1, 0, 0 ],
  [ 0, 0, 1, 0, 0 ],
  [ 0, 1, 0, 1, 1 ],
  [ 0, 0, 1, 0, 0 ],
  [ 0, 0, 1, 0, 0 ]
]
>>>CYCL_COM
ACBDE
Цикломатическую сложность: 7
>>>
```

Рис.1 Пример работы примера

```
>>>ADD_V(S)
>>>ADD_E(C,S,1)
>>>SH_V
[ 'A', 'B', 'C', 'D', 'E', 'S' ]
>>>SH_E
[
  [ 0, 0, 1, 0, 0, 0 ],
  [ 0, 0, 1, 0, 0, 0 ],
  [ 0, 1, 0, 1, 1, 1 ],
  [ 0, 0, 1, 0, 0, 0 ],
  [ 0, 0, 1, 0, 0, 0 ],
  [ 0, 0, 0, 0, 0, 0 ]
]
>>>CYCL_COM
ACBDES
Цикломатическую сложность: 9
>>>
```

Рис.2 Пример работы модифицированного примера

Литература.

1. URL: https://habr.com/ru/post/570612/
2. URL: https://kvodo.ru/incidence-matrix.html
3. URL: https://youtu.be/fCfPjm8u89U