Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

ГРАФЫ

Отчет по лабораторной работе №7 по дисциплине «структуры и алгоритмы обработки данных в ЭВМ»

Обучающийся гр. 439-3
(группа)

_____ А. С. Мазовец
(подпись) *(И. О. Фамилия)*

« _____ » _____ 2021 г.
(дата)

Проверил:

ассистент кафедры АСУ
(должность, ученая степень, звание)

_____ Я. В. Яблонский
(подпись) *(И. О. Фамилия)*

_____ « _____ » _____ 2021 г.
(оценка) *(дата)*

Томск 2021

1 Задание на лабораторную работу

1.1 Вариант 12

Напишите программу, которая будет определять все компоненты двусвязности неориентированного графа. Для представления графа в программе использовать списки смежности. Данные о графе вводятся из файла. После завершения работы с динамическими структурами данных необходимо освободить занимаемую ими память.

2 Алгоритм решения задачи

Сформулируем алгоритм поиска точек сочленения. Реализуем рекурсивнуюфункцию void go(int curr, int prev), где curr — текущая вершина, a prev — вер-шина, из которой мы попали в текущую. При первом вызове curr = r, prev = –1.В теле функции будут выполняться следующие шаги:

1) Запишем в number[curr] номер вершины curr в порядке обхода в глубину.

2) Запишем в L[curr] значение number[curr].

3) Переберем в цикле все вершины, в которые есть ребро из curr. Для каждой такой вершины i выполним следующие действия:

4) Если вершина i еще не посещена, вызовем рекурсивно функцию go с параметрами i, curr. Если после этого значение L[i] стало меньше, чем L[curr],присвоим L[curr] = L[i].

5) Если вершина i уже была посещена и ее номер number[i] < number[curr],и при этом i не равно prev (т.е. ребро (i, prev) обратное и возвращается в верши-ну с меньшим номером), то если L[curr] > number[i], присвоим L[curr] =number[i].

6) Данный алгоритм заполнит массивы L[N] и number[N] требуемым образом. Проверять, является ли вершина точкой сочленения, можно на шаге 3a.Также, если реализовать стек для хранения ребер, можно реализовать вывод са-мих двусвязных компонент: ребро нужно добавлять в стек на шагах 3a (перед рекурсивным вызовом) и 3b и выталкивать из стека в поток вывода все ребра

вплоть до текущего (curr, i), если на шаге 3а выяснилось, что для текущей вершины curr выполняется условие теоремы.

3 Листинг программы

3.1 Основной модуль

```
 1 | from collections import defaultdict
 2 |
 3 | from sys import stdin
 4 | input = stdin.readline
 5 |
 6 |
 7 | class Graph:
 8 |     def __init__ (self, count):
 9 |         self.vertCount  = count
10 |         self.jointsList = defaultdict (set)
11 |         self.iterDFG = 0
12 |
13 |     def addEdge (self, start, end):
14 |         self.jointsList[start].add (end)
15 |         self.jointsList[end].add (start)
16 |
17 |
18 | def biconnect (graph, curr, prev, low, num, stack):
19 |     childs    = 0
20 |     num[curr] = graph.iterDFG
21 |     low[curr] = graph.iterDFG
22 |     graph.iterDFG += 1
23 |
24 |     for i in graph.jointsList[curr]:
25 |         if num[i] == -1 :
26 |             prev[i] = curr
27 |             childs += 1
28 |             stack.append ( (curr, i) )
29 |             biconnect (graph, i, prev, low, num, stack)
30 |
31 |             if (low[i] < low[curr]):
32 |                 low[curr] = low[i]
33 |
34 |             if prev[curr] == -1 and childs > 1 or prev[curr] !
     = -1 and low[i] >= num[curr]:
35 |                 w = -1
36 |                 while w != (curr, i):
37 |                     w = stack.pop ()
38 |                     print (str (w[0]) + " <-> " + str (w[1]),
     end="  ")
39 |                 print ("")
40 |
41 |         elif i != prev[curr] and low[curr] > num[i]:
42 |             low[curr] = min (low [curr], num[i])
```

```
43 |             stack.append ((curr, i))
44 |
45 |
46 | f = open ("I", 'r')
47 | n = int (f.readline ())
48 | g = Graph (n)
49 | for i in range (n):
50 |     a, b = map (int, f.readline ().split ())
51 |     g.addEdge (a, b)
52 | f.close ()
53 |
54 | num     = [-1] * g.vertCount
55 | low     = [-1] * g.vertCount
56 | parent  = [-1] * g.vertCount
57 | stack   = []
58 |
59 | biconnect (g, 1, parent, low, num, stack)
60 | if stack:
61 |     while stack:
62 |         w = stack.pop ()
63 |         print (str (w[-1]) + " <-> " + str (w[1]), end="  ")
64 |     print ("")
```

4 Пример решения задачи

В качестве входного файла выступает текстовый документ I (см. рисунок4.1). Первое число в файле — число ребер графа. Далее идет последовательность пар чисел — начало и конец каждого ребра.

```
asmazovec@mobilehost insert ~/dev/sem4.СиАОДвЭВМ.lab7 > cat I
13
1 5
1 6
2 3
2 6
2 7
3 6
3 8
3 9
4 6
5 6
6 7
8 9
9 10
```

Рисунок 4.1 — входной файл

Результат работы программы выводиться в консоль (см. рисунок 4.2). На-каждой строке выводится отдельная компонента двусвязности в виде спискавхо-дящих в него ребер.

```
asmazovec@mobilehost insert ~/dev/sem4.СиАОДвЭВМ.lab7 > python praph.py
9 ⟷ 10
9 ⟷ 3  8 ⟷ 9  3 ⟷ 8
7 ⟷ 6  2 ⟷ 7  3 ⟷ 6  2 ⟷ 3  6 ⟷ 2
6 ⟷ 4
1 ⟷ 1  6 ⟷ 6  5 ⟷ 5
```

Рисунок 4.2 — результат работы программы

5 Вывод

Был освоен и реализован алгоритм поиска компонент двусвязности, за-креплены знания по теме графы.