Курсовая работа по дискретной математике. 3 Вариант

Галкин Алексей Дмитриевич

1 Задание 8

Нахождение компонент сильной связности графа;

2 Решение

Для начала дадим определение компонентам сильной связности графа. Орграф называется сильно связным, если для любых двух его различных вершин v, w существует путь из v в w. Компонентной связности графа G называется его связный подграф, не являющийся собственным подграфом никакого другого связного подграфа графа G. Компонентной сильной связности орграфа D называется его сильно связный подграф, не являющийся собственным подграфом никакого другого сильно связного подграфа орграфа D.

Для нахождения компонентов сильной связности воспользуемся алгоритмом Косарайю:
1. Запускаем поиск в глубину на исходном графе, запоминая, в каком порядке выходили из вершин.
2. Инвертируем дуги исходного ориентированного графа.
3. Запускаем поиск в глубину на этом обращённом графе,в очередной раз выбирая не посещённую вершину с максимальным номером в векторе, полученном в п.1.
4. Полученные из п.3 деревья и являются сильно связными компонентами.

Инвертирование графа — смена направлений всех рёбер в графе на противоположные
Поиск в глубину (DFS) — алгоритм обхода графа. Алгоритм поиска описывается рекурсивно: перебираем все исходящие из рассматриваемой вершины рёбра. Если ребро ведёт в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать рёбра. Возврат происходит в том случае, если в рассматриваемой вершине не осталось рёбер, которые ведут в нерассмотренную вершину. Если после завершения алгоритма не все вершины

были рассмотрены, то необходимо запустить алгоритм от одной из нерассмотренных вершин.

Для доказательства алгоритма Косарайю приведём несколько Лемм:

1. Если истинно u<>v и v<>w, то истинно u<>w

Док-во: Поскольку u<>v, то v достижима из u, также имеем v<>w, значит w достижима из v. Следовательно w достижима из u, аналогично, u достижима из w. Получаем u<>w.

2. Инвертирование рёбер цикла не влияет на его цикличность

Док-во: Если в исходном графе есть путь из u в v, то в инвертированном графе будет путь из v в u.

Доказательство алгоритма:

Если вершины u и w были сильно связаны в графе G, на третьем этапе будет найден путь из одной вершины в другую, так как на первом шаге был найден путь u→w, а на третьем — путь w→u. Это означает, что по окончании алгоритма обе вершины лежат в одном дереве. Вершины u и w лежат в одном и том же дереве поиска в глубину на третьем шаге алгоритма. Значит, они обе достижимы из корня r этого дерева. Вершина r была рассмотрена на 3 шаге раньше всех, значит время выхода из неё на 1 шаге больше, чем время выхода из вершин u и w. Из этого мы получаем 2 случая:
1. Обе эти вершины были достижимы из r в исходном графе. Это означает сильную связность вершин u и r и сильную связность вершин w и r (по Лемме 2). Склеивая пути, мы получаем связность вершин u и w (по Лемме 1)
2. Хотя бы одна вершина не достижима из r в исходном графе, например w. Значит и r была не достижима из w в исходном графе, так как время выхода из r — больше (если бы она была достижима, то время выхода из v было бы больше, чем из r, просмотрите ещё раз первый шаг примера). Значит между этими вершинами нет пути (ни в исходном, ни в инвертированном графах), но последнего быть не может, потому что по условию w достижима из r на 3 шаге (в инвертированном графе) Значит, из случая 1 и не существования случая 2 получаем, что вершины u и w сильно связаны в обоих графах.

Сложность алгоритма:

Количество ребер в инвертированном равно количеству ребер в изначальном графе, поэтому поиск в глубину будет работать за O(V + E). Если граф будет насыщенным, то есть количество рёбер будет равным n(n-1)/2, где n - кол-во вершин, то сложность будет равна

$$O(V+E) = O(V + \frac{V*(V-1)}{2}) = O(\frac{V^2+V}{2}) = O(V^2)$$

Практическая задача:

Алгоритм можно использовать в разнообразных сферах:

Можно будет спроецировать транспортную сеть на граф, и компоненты

сильной связности покажут, можно ли достичь из окраины в города в центр города. То есть алгоритм поможет в проверке новых транспортных сетей на достижимость.
Потом алгоритмом можно воспользоваться при проектировки вентиляции. Этот алгоритм покажет, нормально циркулирует воздух в воздуховодов или нет.

Код программы на языке Python с использованием библиотек networkx и matplotlib.pyplot:
import math
import networkx as nx
import matplotlib.pyplot as plt

def Input$_{G}raph(Matrix_{o}f_{A}djacency, n):$
$graph = nx.DiGraph()$
$foriinrange(n):$
$forjinrange(n):$
$if(Matrix_{o}f_{A}djacency[i][j] > 0):$
$graph.add_{n}odes_{f}rom([i, j])$
$graph.add_{e}dge(i, j)$
$returngraph$

$defPrint_{G}raph(graph : nx.DiGraph):$
$labels = dict()$
$fornodeingraph.nodes:$
$labels[node] = node + 1$
$pos = nx.spring_{l}ayout(graph)$
$nx.draw(graph, pos, with_{l}abels = True, labels = labels)$
$nx.draw(graph, pos)$
$plt.show()$

$defEnter_{M}atrix(Matrix_{o}f_{A}djacency, n, g, gr):$
$k = 0$
$foriinrange(n):$
$Matrix_{o}f_{A}djacency.append(list(map(int, input().split())))$
$mat = Matrix_{o}f_{A}djacency[i]$
$forjinrange(n):$
$ifmat[j] == 1:$
$g[k].append(j)$
$gr[j].append(k)$
$k+ = 1$

$defAlgorithm(n, g, gr):$
$tout = []$
$visited = [Falseforiinrange(n)]$
$foriinrange(n):$
$ifnotvisited[i]:$

$dfs1(i, visited, g, tout)$

$visited = [Falseforiinrange(n)]$
$tout = tout[:: -1]$
$forvintout :$
$scc = []stronglyconnectedcompoents$
$ifnotvisited[v] :$
$dfs2(v, scc, visited, gr)$
$scc.reverse()$
$forjinrange(len(scc)) :$
$scc[j] + = 1$
$print(scc)$

$defdfs1(v, visited, g, tout) :$
$visited[v] = True$
$foruing[v] :$
$ifnotvisited[u] :$
$dfs1(u, visited, g, tout)$
$tout.append(v)$

def dfs2(v, scc, visited, gr):
visited[v] = True
scc.append(v)
for u in gr[v]:
if not visited[u]:
dfs2(u,scc, visited, gr)

def main ():
print ("Введите количество вершин графа: end=)
n = int (input ())
g = [[] for i in range(n)]
gr = [[] for i in range(n)]
Matrix$_o f_A djacency = []$
$print(" : ")$
$Enter_M atrix(Matrix_o f_A djacency, n, g, gr)$
$print(" : ")$
$Algorithm(n, g, gr)$
$graph = Input_G raph(Matrix_o f_A djacency, n)$
$Print_G raph(graph)$
$print()$

if ${}_{n}ame_{=="}{}_{main_{":main()}}$