Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №9

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний в графе»

Выполнили:

студенты группы 22BВВ1

Митрошин Ю.Е

Коннов А.Д

Приняли:

Акифьев И.В

Юрова О.В

Пенза 2023

**Название**

Поиск расстояний в графе

**Общие сведения.**

Поиск расстояний – довольно распространенная задача анализа графов.

Для поиска расстояний можно использовать процедуры обхода графа. Для этого при каждом переходе в новую вершину необходимо запоминать, сколько шагов до нее мы сделали. При этом вектор, который хранил информацию о посещении вершин становится вектором расстояний. Довольно просто модернизировать для поиска расстояний в графе алгоритм обхода в ширину, т.к. этот алгоритм проходит вершины по уровням удаленности, то для не ориентированного графа для вершин каждого следующего уровня глубины расстояние от исходной вершины увеличивается на 1. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины.

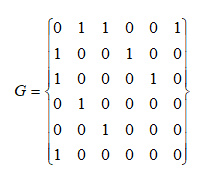
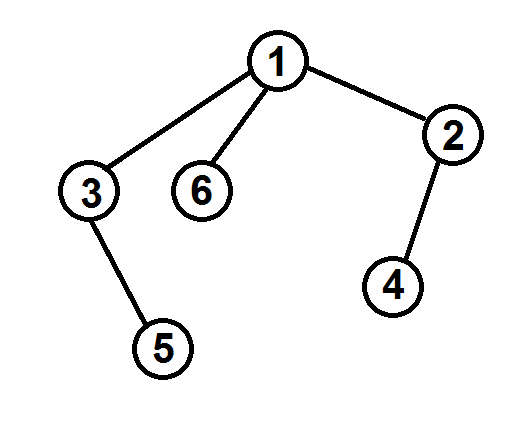


Рисунок 1 – Граф

Таким образом, можно предложить следующую реализацию алгоритма обхода в ширину.

**Вход**: G – матрица смежности графа, v – исходная вершина.

**Выход**: DIST – вектор расстояний до всех вершин от исходной.

**Алгоритм ПОШ**

1.1. для всех i положим DIST [i] =  -1 пометим как "не посещенную";

1.2. **ВЫПОЛНЯТЬ** BFSD (v).

1.3  для всех i вывести DIST [i] на экран;

**Алгоритм** BFSD(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. Обновить вектор расстояний  DIST [ x ] = 0;

2.4. **ПОКА**  Q != ∅ очередь не пуста **ВЫПОЛНЯТЬ**

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. **ДЛЯ** i = 1 **ДО**size\_G**ВЫПОЛНЯТЬ**

2.9.   **ЕСЛИ**  G(v,i) = = 1**И** DIST = = -1

2.10. **ТО**

2.11.       Поместить i в очередь Q.push(i);

2.12.       Обновить вектор расстояний DIST [ i ] = DIST [ v ] + 1;

Реализация состоит из подготовительной части, в которой все вершины помечаются как не поcещенные (п.1.1). В отличие от алгоритма BFS не посещенные вершины помечаем -1, т.к. значение 0 и 1 могут быть расстояниями. Расстояние 0 – от исходной вершины до самой себя.

В самой процедуре как и в алгоритме BFS сначала создается пустая очередь (п. 2.1), в которую помещается исходная вершина, из которой начат обход (п.2.2). Расстояние до этой вершины (п.2.3) устанавливается равным 0 (расстояние до самой себя).

Далее итерационно, пока очередь не опустеет, из нее извлекается первый элемент, который становится текущей вершиной (п. 2.5, 2.6). Затем в цикле просматривается **v**-я строка матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с **v** не посещенную вершину (п.2.9), эта вершина помещается в очередь (п.2.11) и для нее обновляется вектор расстояния (п.2.12). Расстояние до новой **i**-й вершины вычисляется как расстояние до текущей **v**-й вершины плюс 1 (так как ребра нашего графа не взвешенные).

После просмотра строки матрицы смежности алгоритм делает следующую итерацию цикла 2.4 или заканчивает работу, если очередь пуста.

Таким образом, если вершина помещается в очередь при просмотре сроки матрицы смежности на 1-й итерации, то они находятся на 1 уровне удаленности и расстояние до этих вершин будет равным 1.

DIST [ i ] = DIST [ v ] + 1, где DIST [ v ] = 0 – расстояние от исходной вершины до самой себя.

Далее, начинают просматриваться вершины первого уровня и соответствующие им строки матрицы смежности. При добавлении смежных с вершинами первого уровня вершин, расстояния до них будут равны 2.

DIST [ i ] = DIST [ v ] + 1, где DIST [ v ] = 1 – расстояние от исходной вершины до вершин 1 уровня.

После того, как все вершины первого уровня будут просмотрены и извлечены из очереди, начнется просмотр вершин 2 уровня и соответствующих им строк матрицы смежности. При добавлении смежных с вершинами второго уровня вершин, расстояния до них будут равны 3.

DIST [ i ] = DIST [ v ] + 1, где DIST [ v ] = 2 – расстояние от исходной вершины до вершин 2 уровня.

И так далее, алгоритм проходит вершины по уровням, пока очередь не опустеет.

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При  реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

**Задание 2\***

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.
2. Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

**Листинг**

from queue import Queue

import random

import time

size1 = int(input("Введитеразмерматрицы M1: "))

M1 = []

start\_vertex = 0

for i in range(size1):

row = []

for j in range(size1):

row.append(0)

M1.append(row)

for i in range(size1):

for j in range(i + 1, size1):

M1[i][j] = M1[j][i] = random.randint(0, 1)

lists = [[] for \_ in range(size1)]

for i in range(size1):

for j in range(size1):

if M1[i][j] == 1:

lists[i].append(j)

defbreadth\_first\_search\_distance(G, v):

size\_G = len(G)

DIST = [-1] \* size\_G

def BFSD(start\_vertex):

Q = Queue()

Q.put(start\_vertex)

DIST[start\_vertex] = 0

while not Q.empty():

current\_vertex = Q.get()

print(current\_vertex, end=' ')

for i in range(size\_G):

if G[current\_vertex][i] == 1 and DIST[i] == -1:

Q.put(i)

DIST[i] = DIST[current\_vertex] + 1

BFSD(v)

return DIST

defbreadth\_first\_search\_distance\_list(adj\_list, start\_vertex):

num\_vertices = len(adj\_list)

DIST = [-1] \* num\_vertices

def BFSD(start):

queue = Queue()

queue.put(start)

DIST[start] = 0

while not queue.empty():

current\_vertex = queue.get()

print(current\_vertex, end=' ')

for neighbor in adj\_list[current\_vertex]:

if DIST[neighbor] == -1:

queue.put(neighbor)

DIST[neighbor] = DIST[current\_vertex] + 1

BFSD(start\_vertex)

return DIST

defdepth\_first\_search\_distance(G):

defDFS(v, distance):

NUM[v] = True

DIST[v] = distance

print(v, end=' ')

for i in range(size\_G):

if G[v][i] == 1 and not NUM[i]:

DFS(i, distance + 1)

size\_G = len(G)

NUM = [False] \* size\_G

DIST = [-1] \* size\_G

for i in range(size\_G):

if not NUM[i]:

DFS(i, 0)

return DIST

defdepth\_first\_search\_distance\_list(adj\_list):

num\_vertices = len(adj\_list)

DIST = [-1] \* num\_vertices

defDFS(vertex, distance):

NUM[vertex] = True

DIST[vertex] = distance

print(vertex, end=' ')

for neighbor in adj\_list[vertex]:

if not NUM[neighbor]:

DFS(neighbor, distance + 1)

NUM = [False] \* num\_vertices

for vertex in range(num\_vertices):

if not NUM[vertex]:

DFS(vertex, 0)

return DIST

print("Матрицасмежностидля M1:")

for row in M1:

print(row)

print("\nСписок смежности для M1: ")

for i, verticles in enumerate(lists):

print(f"Вершина {i}: {verticles}")

print("\nРезультат обхода в ширину с расстояниями:")

start\_time = time.time()

distances = breadth\_first\_search\_distance(M1, start\_vertex)

elapsed\_time = time.time() - start\_time

print("\nВекторрасстоянийотвершины", start\_vertex, ":", distances)

print(f"Времяработы: {elapsed\_time} секунд")

print("\nРезультат обхода в ширину с расстояниями для списка смежности:")

distances = breadth\_first\_search\_distance\_list(lists, start\_vertex)

print("\nВекторрасстоянийотвершины", start\_vertex, ":", distances)

print("\nРезультат обхода в глубину с расстояниями:")

start\_time = time.time()

distances = depth\_first\_search\_distance(M1)

elapsed\_time = time.time() - start\_time

print("\nВекторрасстояний:", distances)

print(f"Времяработы: {elapsed\_time} секунд")

print("\nРезультат обхода в глубину с расстояниями для списка смежности:")

distances = depth\_first\_search\_distance\_list(lists)

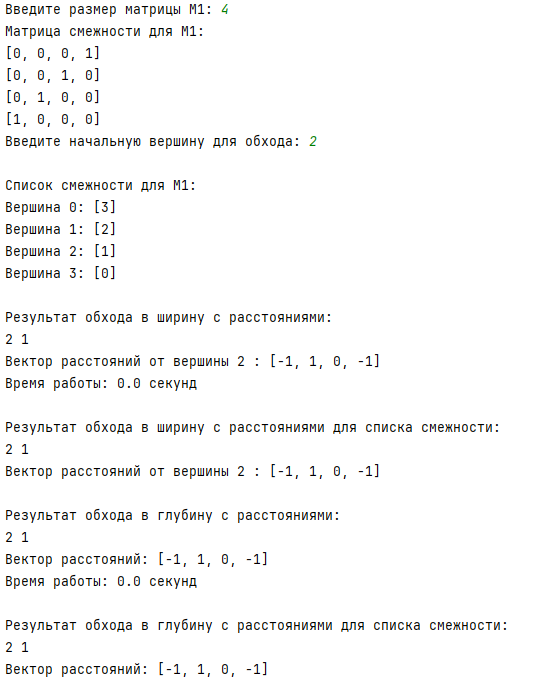
print("\nВектор расстояний:", distances)

**Анализ работы программы:**

Быстрее работает обход в глубину, это отображено в таблице

|  |  |  |
| --- | --- | --- |
| Размер матрицы | Время обхода в ширину, с | Время обхода в глубину, с |
| 100 | 0.00201 | 0.00199 |
| 500 | 0.01912 | 0.01708 |
| 1000 | 0.09182 | 0.08510 |
| 2000 | 0.33435 | 0.30991 |

**Результат работы программы:**

****

**Вывод:** в ходе лабораторной работы мы научились производить поиск расстояния в графе.