**Дисциплина: Методы оптимизации**

**Раздел: Потоки в сетях**

**Автор: Филиппова Анна Сергеевна, кафедра ПИ, БГПУ им.М.Акмуллы Обязательно к изучению по направлению 09.03.03 Прикладная информатика (уровень бакалавриат)**

**Студент: Нурализода Исломиддин**

**Группа : ПИНФ\_ПРЦ\_31-22**

**ЛАБОРАТОРНАЯ РАБОТА №2.**

**КРАТЧАЙШИЕ ПУТИ МЕЖДУ ВСЕМИ ПАРАМИ УЗЛОВ (ПРОГРАММНЫЙ МОДУЛЬ)**

**Цель и задачи лабораторной работы**

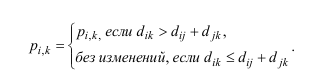
Цель работы – изучить алгоритм с тройственными операциями для поиска кратчайшего пути между всеми парами узлов в сети.

Задачи: реализовать алгоритм для поиска кратчайшего пути между всеми парами узлов в сети.

**Теоретическая часть:**

**Кратчайшие пути между всеми парами узлов**

В большинстве приложений с ребрами или вершинами ассоциируются некоторые числа. В этом случае граф называется сетью. Все определения теории графов применимы и к сетям. В теории сетей обычно используют термины «узлы» и «дуги» вместо «вершины» и «ребра». Длина пути есть сумма длин всех его дуг. Обычно имеется много путей между двумя вершинами, путь минимальной длины называется кратчайшим путем. Задача нахождения кратчайшего пути является фундаментальной и часто входит как подзадача в другие оптимизационные задачи. Рассмотрим задачу поиска кратчайших путей между всеми парами узлов сети. e ij e ik e kl e pq Пусть , , , …, , - кратчайший путь из Vi в Vq. Тогда кратчайший путь из Vi в Vj должен представлять собой единственную дугу еij , кратчайший путь из Vj в Vk — дугу ejk, и т. д. Назовем дугу еij базисной, если она представляет собой кратчайший путь из Vi в Vj. Из данного определения следует, что кратчайший путь состоит только из базисных дуг. Алгоритм, который мы опишем, заменяет все небазисные дуги базисными. Т.е., алгоритм строит дуги, соединяющие каждую пару узлов, не соединенную базисной дугой. Длина каждой построенной дуги равна кратчайшему расстоянию между двумя узлами. Для данного узла Vj рассмотрим следующую простую операцию: d ik ¬ min { d ik , d ij + d jk } (1) Операция выполняется для каждого фиксированного j и всевозможных i и k, не равных j. Для трех узлов Vj, Vj и Vk и трех дуг с длинами dik , dij и djk данная операция сравнивает длину дуги еik , с длиной пути, состоящего из двух дуг, с промежуточным узлом Vj. Операция (1) называется тройственной операцией. Для всевозможных пар узлов Vi и Vk, смежных с Vj , выполним следующие операции: £- если dik dij + djk, то никаких действий не производим; - если dik > dij + djk,, то создаем новую дугу, ведущую из Vi в Vk с dik = d ij + d jk . Схема алгоритма: 1) Фиксируем j = 1 и выполняем тройственную операцию (1) для всех i, k = 2, 3,..., n. 2) Фиксируем j = 2 и выполняем тройственную операцию (1) для всех i, k = 1, 3,..., n. Замечание. Все новые дуги, добавленные при j = 1, используются далее при j = 2, и т.д. Кратчайшие расстояния между каждой парой узлов найдены. Но необходимо найти промежуточные (внутренние) узлы для кратчайших путей. Чтобы зафиксировать порядок, в котором появляются эти узлы, мы используем матрицу [pi,k], в которой элемент i-ой строки и k-ro столбца указывает на первый внутренний узел в пути из Vi в Vj,. Если Рik = j, то кратчайший путь имеет вид Vi, Vj,..., Vk. Далее, если pj,k = s, то кратчайший путь имеет вид Vi,Vj, Vs,..., Vk. Изначально мы полагаем Pi,k = k для всех i, k. Таким образом, предполагается, что каждая дуга — базисная (до тех пор, пока не установлено обратное), и первым внутренним узлом на пути из Vi в Vk является сам Vk. Во время выполнения тройственной операции мы также обновляем данные в таблице промежуточных узлов. Элементы таблицы промежуточных узлов изменяются согласно следующему правилу:



**Варианты заданий**

**Вариант 1.**

Кратчайшие пути между всеми парами узлов. Взвешенный граф, число вершин – 15.

Выход: Кратчайшее расстояние и путь.

На язык python

import numpy as np

def triple\_operation(graph):

    n = len(graph)

    dist = np.copy(graph)  # Матрица расстояний

    path = np.zeros((n, n), dtype=int)  # Матрица промежуточных узлов

    # Инициализация матрицы промежуточных узлов

    for i in range(n):

        for k in range(n):

            if i != k and dist[i][k] != np.inf:

                path[i][k] = k

            else:

                path[i][k] = -1

    # Тройственная операция

    for j in range(n):

        for i in range(n):

            if i == j:

                continue

            for k in range(n):

                if k == j:

                    continue

                if dist[i][j] + dist[j][k] < dist[i][k]:

                    dist[i][k] = dist[i][j] + dist[j][k]

                    path[i][k] = path[i][j]

    return dist, path

def get\_shortest\_path(path, i, k):

    if path[i][k] == -1:

        return []

    result = [i]

    while i != k:

        i = path[i][k]

        result.append(i)

    return result

# Пример использования

graph = [

    [0, 3, np.inf, 7],

    [8, 0, 2, np.inf],

    [5, np.inf, 0, 1],

    [2, np.inf, np.inf, 0]

]

dist, path = triple\_operation(graph)

print("Матрица кратчайших расстояний:")

print(dist)

print("\nМатрица промежуточных узлов:")

print(path)

# Пример поиска пути

start = 0

end = 3

shortest\_path = get\_shortest\_path(path, start, end)

print(f"\nКратчайший путь из {start} в {end}: {shortest\_path}")

Ответ

