САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3

по курсу «Алгоритмы и структуры данных»

Тема: Графы

Вариант 22

Выполнил:

Федюкин М. В.

К3244

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

Содержание отчета

Задачи по варианту

Задание к лабораторной работе № 2-3:
https://drive.google.com/drive/folders/1hjwL6oDXaZJ8BZqDXJec6fxwhDpgdbmX

Мой вариант – 22.

Обязательные задачи: 2, 7, 16.

Дополнительные задачи: 1, 3, 4, 5, 6, 8, 11, 12. 13, 15.

Обязательные задачи

Задача 2

Задача во многом аналогична предыдущей.
Вводим глобальную переменную-счетчик, которая увеличивается всякий раз, когда мы начинаем путь из еще не посещенной вершины. Нужно отметить, что это рекурсивный алгоритм, он ест много памяти, хотя для имеющихся на задачу ограничений это не критично. Абсолютно этот же алгоритм, но в итеративной форме, представлен в задаче 13.

```
from test import time_memory
import threading


def explore(graph, visited, curr_key):
    visited[curr_key] = True
    for q in graph[curr_key]:
        if not visited[q]:
            explore(graph, visited, q)


def main():
    with open('input.txt', 'r') as infile:
        n, m = map(int, infile.readline().split())
        graph = {}
        visited = {}
        for i in range(1, n+1):
            graph[i] = set()
            visited[i] = False
        for _ in range(m):
            u, v = map(int, infile.readline().split())
            graph[u].add(v)
            graph[v].add(u)
    counter = 0
    for q in graph.keys():
        if not visited[q]:
            counter += 1
            explore(graph, visited, q)
    with open('output.txt', 'w') as outfile:
        print(counter, file=outfile)


if __name__ == '__main__':
    thread = threading.Thread(target=time_memory(main))
    thread.start()
```

Input	Output
4 2 1 2 3 2	2
6 2 1 6 2 4	4
5 7 2 1 3 2 3 1 4 3 4 1 5 2 5 3	1

Задача 7

Пусть каждая вершина имеет цвет (изначально никакой) и метку, посещена ли она. Когда мы обходим очередную вершину, то отмечаем её как посещенную и красим всех её детей в противоположный цвет. Если из текущей вершины мы попадаем в уже посещенную вершину такого же цвета, как текущая, значит, граф не двудольный.

```
from collections import deque
from test import time_memory
import threading


def inverted(n):
    return 1 if n == 0 else 0


def main():
    with open('input.txt', 'r') as infile:
        n, m = map(int, infile.readline().split())
        graph = {}
        for i in range(1, n+1):
            graph[i] = {"kids": set(),
                        "visited": False,
                        "color": -1}
        for _ in range(m):
            u, v = map(int, infile.readline().split())
            graph[u]["kids"].add(v)
            graph[v]["kids"].add(u)
    flag = True
    graph[1]["color"] = 0
    queue = deque()
    queue.append(1)
    while len(queue) > 0:
        u = queue.popleft()
        graph[u]["visited"] = True
        for v in graph[u]["kids"]:
            if not graph[v]["visited"]:
                graph[v]["color"] = inverted(graph[u]["color"])
                queue.append(v)
            else:
                if graph[v]["color"] !=
inverted(graph[u]["color"]):
                    flag = False
                    break
    with open('output.txt', 'w') as outfile:
        print(1 if flag else 0, file=outfile)


if __name__ == '__main__':
    thread = threading.Thread(target=time_memory(main))
    thread.start()
```

Input	Output
4 4 1 2 4 1 2 3 3 1	0
5 4 5 2 4 2 3 4 1 4	1
6 5 1 6 3 2 2 4 2 5 4 5 3 5	1

Задача 16

Задача сводится к поиску цикла в графе. Просто делаю обход графа, если натыкаюсь на уже пройденную вершину – значит, цикл есть. Единственное, я переписала рекурсивный алгоритм, использовавшийся в одной из первых задач, на итеративный.

```
from collections import deque
from test import time_memory
import threading


def cycle(graph, start_key):
    queue = deque()
    queue.append(start_key)
    while len(queue) > 0:
        curr_key = queue.pop()
        graph[curr_key]["visited"] = True
        for key in graph[curr_key]["kids"]:
            if key == start_key:
                return True
            if not graph[key]["visited"]:
                queue.append(key)
    return False


def main():
    n = int(input())
    graph = {}
    for _ in range(n):
        key = input()
        graph[key] = {"kids": set(),
                      "visited": False}
        m = int(input())
        for _ in range(m):
            kid = input()
            graph[key]["kids"].add(kid)
        input()
    for start_key in graph.keys():
        flag = True
        for key in graph.keys():
            graph[key]["visited"] = False
        print("YES" if cycle(graph, start_key) else "NO")


if __name__ == '__main__':
    thread = threading.Thread(target=time_memory(main))
    thread.start()
```

19898975	19.09.2023 19:33:58	Федюкин Михаил	0345	Python	Accepted		0,109	2746 Кб

Дополнительные задачи

Задача 1

Если мы достигли из стартовой вершины искомую, возвращаю положительный ответ и прекращаю обход, иначе обхожу граф до конца и возвращаю отрицательный ответ. Сам граф задан словарем, посещенные вершины тоже заданы словарем.

```
from test import time_memory
import threading

found = False


def explore(graph, visited, curr_key, goal):
    global found
    if curr_key == goal:
        found = True
        return
    visited[curr_key] = True
    for key in graph[curr_key]:
        if not visited[key]:
            explore(graph, visited, key, goal)


def main():
    with open('input.txt', 'r') as infile:
        n, m = map(int, infile.readline().split())
        graph = {}
        visited = {}
        for i in range(1, n+1):
            graph[i] = set()
            visited[i] = False
        for _ in range(m):
            u, v = map(int, infile.readline().split())
            graph[u].add(v)
            graph[v].add(u)
        u, v = map(int, infile.readline().split())
    explore(graph, visited, u, v)
    with open('output.txt', 'w') as f:
        print(1 if found else 0, file=f)


if __name__ == '__main__':
    thread = threading.Thread(target=time_memory(main))
    thread.start()
```

Input	Output
4 4 1 2 3 2 4 3 1 4 1 4	1
4 2 1 2 3 2 1 4	0
5 7 2 1 3 2 3 1 4 3 4 1 5 2 5 3 2 5	1

Задача 3

Если при текущем обходе мы наткнулись на вершину, в которую уже заходили, значит имеется цикл. Важный момент: при каждом новом обходе (т.е. из новой вершины) нужно начинать с чистого листа – обнулять уже посещенные вершины.

```
from test import time_memory
import threading


def explore(graph, visited, curr_key):
    visited[curr_key] = True
    for key in graph[curr_key]:
        if visited[key]:
            return False
        return explore(graph, visited, key)
    return True


def main():
    with open('input.txt', 'r') as infile:
        n, m = map(int, infile.readline().split())
        graph = {}
        visited = {}
        for i in range(1, n+1):
            graph[i] = set()
            visited[i] = False
        for _ in range(m):
            u, v = map(int, infile.readline().split())
            graph[u].add(v)
    flag = False
    for key in graph.keys():
        for i in range(1, n + 1):
            visited[i] = False
        outcome = explore(graph, visited, key)
        if not outcome:
            flag = True
            break
    with open('output.txt', 'w') as outfile:
        print(1 if flag else 0, file=outfile)


if __name__ == '__main__':
    thread = threading.Thread(target=time_memory(main))
    thread.start()
```

Input	Output
4 4 1 2 4 1 2 3 3 1	1
5 7 1 2 2 3 1 3 3 4 1 4 2 5 3 5	0
5 7 2 1 3 2 3 1 4 3 1 4 5 2 5 3	1

Задача 4

В данной задаче реализован алгоритм топологической сортировки из лекции.

```
from test import time_memory
import threading

clock = 0


def explore(graph, curr_key):
    global clock
    graph[curr_key]["visited"] = True
    graph[curr_key]["pre"] = clock
    clock += 1
    for key in graph[curr_key]["kids"]:
        if not graph[key]["visited"]:
            explore(graph, key)
    graph[curr_key]["post"] = clock
    clock += 1


def main():
    with open('input.txt', 'r') as infile:
        n, m = map(int, infile.readline().split())
        graph = {}
        for i in range(1, n+1):
            graph[i] = {"kids": set(),
                        "visited": False,
                        "pre": None,
                        "post": None}
        for _ in range(m):
            u, v = map(int, infile.readline().split())
            graph[u]["kids"].add(v)
    for key in graph.keys():
        if not graph[key]["visited"]:
            explore(graph, key)
    with open('output.txt', 'w') as outfile:
        for para in sorted(graph.items(), key=lambda para:
1/para[1]["post"]):
            print(para[0], end=" ", file=outfile)


if __name__ == '__main__':
    thread = threading.Thread(target=time_memory(main))
    thread.start()
```

Input	Output
4 3 1 2 4 1 3 1	4 3 1 2
4 1 3 1	4 3 2 1
5 7 2 1 3 2 3 1 4 3 4 1 5 2 5 3	5 4 3 2 1

Задача 5

Реализация данного алгоритма взята из лекции. Сначала делаю топологическую сортировку инвертированного графа. Затем для вершин-стоков инвертированного графа обращаюсь к исходному графу и делаю обход.

```
from test import time_memory
import threading

clock = 0


def explore(graph, curr_key):
    global clock
    graph[curr_key]["visited"] = True
    graph[curr_key]["pre"] = clock
    clock += 1
    for key in graph[curr_key]["kids"]:
        if not graph[key]["visited"]:
            explore(graph, key)
    graph[curr_key]["post"] = clock
    clock += 1


def main():
    with open('input.txt', 'r') as infile:
        n, m = map(int, infile.readline().split())
        graph = {}
        graph_t = {}
        for i in range(1, n+1):
            graph[i] = {"kids": set(),
                        "visited": False,
                        "pre": None,
                        "post": None}
            graph_t[i] = {"kids": set(),
                          "visited": False,
                          "pre": None,
                          "post": None}
        for _ in range(m):
            u, v = map(int, infile.readline().split())
            graph[u]["kids"].add(v)
            graph_t[v]["kids"].add(u)
    for key in graph_t.keys():
        if not graph_t[key]["visited"]:
            explore(graph_t, key)
    scc = 0
    for para in sorted(graph_t.items(), key=lambda para: 1 /
para[1]["post"]):
        if not graph[para[0]]["visited"]:
            scc += 1
            explore(graph, para[0])
```

```
    with open('output.txt', 'w') as outfile:
        print(scc, file=outfile)


if __name__ == '__main__':
    thread = threading.Thread(target=time_memory(main))
    thread.start()
```

Input	Output
4 4 1 2 4 1 2 3 3 1	2
5 7 2 1 3 2 3 1 4 3 4 1 5 2 5 3	5
4 2 3 1 1 3	3

Задача 6

В данной задаче реализован алгоритм Дейкстры. Всегда выбираем вершину с наименьшим весом и проходимся по всем ее детям, изменяя их вес, если это дает положительный результат.

```
from test import time_memory
import threading
from collections import deque


def main():
    with open('input.txt', 'r') as infile:
        n, m = map(int, infile.readline().split())
        graph = {}
        for i in range(1, n+1):
            graph[i] = {"kids": set(),
                        "path": float("inf")}
        for _ in range(m):
            u, v = map(int, infile.readline().split())
            graph[u]["kids"].add(v)
            graph[v]["kids"].add(u)
        start, end = map(int, infile.readline().split())
    graph[start]["path"] = 0
    queue = deque()
    queue.append(start)
    while len(queue) > 0:
        u = queue.popleft()
        for v in graph[u]["kids"]:
            if graph[v]["path"] == float("inf"):
                queue.append(v)
                graph[v]["path"] = graph[u]["path"] + 1
    with open('output.txt', 'w') as outfile:
        print(graph[end]["path"] if graph[end]["path"] !=
float("inf") else -1, file=outfile)


if __name__ == '__main__':
    thread = threading.Thread(target=time_memory(main))
    thread.start()
```

Input	Output
4 4 1 2 4 1 2 3 3 1 2 4	2
5 4	-1

5 2 1 3 3 4 1 4 3 5	
6 5 1 6 3 2 2 4 2 5 4 5 3 5	2

Задача 8

Тот же алгоритм Дейкстры, только в отличие от 6-ой задачи вес ребра задан, а не равен единице, и он складывается в матрицу смежности.

```
from test import time_memory
import threading


def get_smallest(dictionary):
    key = -1
    mn = float("inf")
    for curr_key in dictionary.keys():
        if not dictionary[curr_key]["visited"] and
dictionary[curr_key]["path"] < mn:
            mn = dictionary[curr_key]["path"]
            key = curr_key
    return key


def main():
    with open('input.txt', 'r') as infile:
        n, m = map(int, infile.readline().split())
        matrix = [[-1 for j in range(m)] for i in range(n)]
        graph = {}
        for i in range(n):
            graph[i] = {"kids": set(),
                        "path": float("inf"),
                        "visited": False}
        for _ in range(m):
            u, v, l = map(int, infile.readline().split())
            u, v = u-1, v-1
            matrix[u][v] = l
            graph[u]["kids"].add(v)
        s, f = map(int, infile.readline().split())
        s, f = s-1, f-1
    graph[s]["path"] = 0
    smallest = s
    while smallest != -1:
        for key in graph[smallest]["kids"]:
            graph[key]["path"] = min(graph[smallest]["path"] +
matrix[smallest][key], graph[key]["path"])
        graph[smallest]["visited"] = True
        smallest = get_smallest(graph)
    with open("output.txt", "w") as outfile:
        print(graph[f]["path"] if graph[f]["path"] !=
float("inf") else -1, file=outfile)


if __name__ == '__main__':
    thread = threading.Thread(target=time_memory(main))
    thread.start()
```

Input	Output
4 4 1 2 1 4 1 2 2 3 2 1 3 5 1 3	3
5 9 1 2 4 1 3 2 2 3 2 3 2 1 2 4 2 3 5 4 5 4 1 2 5 3 3 4 4 1 5	6
3 3 1 2 7 1 3 5 2 3 2 3 2	-1

Задача 11

В данной задаче необходимо построить граф и применить какой-нибудь из обходов, чтобы выяснить, существует ли путь из одной вершины в другую. Также необходимо обработать ситуации, когда исходная и конечная вершина отсутствуют в построенном графе.

```
from collections import deque


if __name__ == "__main__":
    n = int(input())
    graph = {}
    for _ in range(n):
        one, symbol, two = input().split()
        try:
            graph[one]["kids"].append(two)
        except KeyError:
            graph[one] = {"kids": [two],
                          "path": float("inf")}
    start = input()
    end = input()
    try:
        graph[start]["path"] = 0
    except KeyError:
        graph[start] = {"kids": [],
                        "path": 0}
    queue = deque()
    queue.append(start)
    while len(queue) > 0:
        u = queue.popleft()
        for v in graph[u]["kids"]:
            try:
                if graph[v]["path"] == float("inf"):
                    queue.append(v)
                    graph[v]["path"] = graph[u]["path"] + 1
            except KeyError:
                graph[v] = {"kids": [],
                            "path": graph[u]["path"] + 1}
    try:
        print(graph[end]["path"] if graph[end]["path"] !=
float("inf") else -1)
    except KeyError:
        print(-1)
```

19899167	19.09.2023 19:58:44	Федюкин Михаил	0743	Python	Accepted		0,046	586 Кб

Задача 12

Это очень простая задача, для решения которой даже не нужны графы. Достаточно задать словарь со словарями и потом пройтись по нему.

```python
if __name__ == "__main__":
    n, m = map(int, input().split())
    maze = {}
    for i in range(1, n+1):
        maze[i] = {}
    for _ in range(m):
        u, v, c = map(int, input().split())
        maze[u][c] = v
        maze[v][c] = u
    n = input()
    path = list(map(int, input().split()))
    curr = 1
    flag = True
    for step in path:
        try:
            curr = maze[curr][step]
        except KeyError:
            flag = False
            break
    print("INCORRECT" if not flag else curr)
```

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
19899182	19.09.2023 19:59:59	Федюкин Михаил	0601	Python	Accepted		0.109	12 Мб

Задача 13

Задача аналогична второй (поиск компонент графа). Но рекурсивный алгоритм изменен на итеративный.

```
from collections import deque


if __name__ == "__main__":
    n, m = map(int, input().split())
    matrix = []
    for _ in range(n):
        matrix.append(input())
    graph = {}
    for i in range(n):
        for j in range(m):
            if matrix[i][j] == "#":
                graph[i * m + j] = {"kids": set(),
                                    "visited": False}
    for i in range(n):
        for j in range(m):
            if matrix[i][j] != "#":
                continue
            key = i * m + j
            neighbours = []
            if i-1 >= 0 and matrix[i-1][j] == "#":
                neighbours.append((i-1) * m + j)
            if i+1 < n and matrix[i+1][j] == "#":
                neighbours.append((i+1) * m + j)
            if j-1 >= 0 and matrix[i][j-1] == "#":
                neighbours.append(i * m + (j-1))
            if j+1 < m and matrix[i][j+1] == "#":
                neighbours.append(i * m + (j+1))
            for v in neighbours:
                graph[key]["kids"].add(v)
                graph[v]["kids"].add(key)
    queue = deque()
    counter = 0
    for overall_key in graph.keys():
        if not graph[overall_key]["visited"]:
            counter += 1
            queue.append(overall_key)
            while len(queue) > 0:
                curr_key = queue.pop()
                graph[curr_key]["visited"] = True
                for key in graph[curr_key]["kids"]:
                    if not graph[key]["visited"]:
                        queue.append(key)
    print(counter)
```

19899188	19.09.2023 20:01:50	Федюкин Михаил	0432	Python	Accepted		0.187	11 Мб

Задача 15

Задача аналогична шестой, отличается только формат ввода графа. Я действую схоже с 13 задачей – каждая клетка матрицы является вершиной графа.

```
from collections import deque


if __name__ == "__main__":
    n, m = map(int, input().split())
    matrix = []
    for _ in range(n):
        matrix.append(input())
    graph = {}
    for i in range(n):
        for j in range(m):
            if matrix[i][j] == "0":
                graph[i * m + j] = {"kids": set(),
                                    "path": float("inf")}
    for i in range(n):
        for j in range(m):
            if matrix[i][j] != "0":
                continue
            key = i * m + j
            neighbours = []
            if i-1 >= 0 and matrix[i-1][j] == "0":
                neighbours.append((i-1) * m + j)
            if i+1 < n and matrix[i+1][j] == "0":
                neighbours.append((i+1) * m + j)
            if j-1 >= 0 and matrix[i][j-1] == "0":
                neighbours.append(i * m + (j-1))
            if j+1 < m and matrix[i][j+1] == "0":
                neighbours.append(i * m + (j+1))
            for v in neighbours:
                graph[key]["kids"].add(v)
                graph[v]["kids"].add(key)
    q_s, q_e, time = map(int, input().split())
    start_key = (q_s-1)*m + q_e-1
    boys = []
    for _ in range(4):
        s, e, k = map(int, input().split())
        key = (s-1)*m + e-1
        boys.append((key, k))
    graph[start_key]["path"] = 0
    queue = deque()
    queue.append(start_key)
    while len(queue) > 0:
        u = queue.popleft()
        for v in graph[u]["kids"]:
            if graph[v]["path"] == float("inf"):
                queue.append(v)
```

```
                graph[v]["path"] = graph[u]["path"] + 1
    count = 0
    for boy in boys:
        if graph[boy[0]]["path"] <= time:
            count += boy[1]
    print(count)
```

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
19899206	19.09.2023 20:04:51	Федюкин Михаил	0846	Python	Accepted		0,046	578 Кб

Вывод

Было сложно, а потом попроще, а потом снова сложно, и на конец наступил конец.
Главное, чему я научился за эту работу – использовать очередь для замены рекурсии итеративным алгоритмом.