

Университет ИТМО

Инфокоммуникационные технологии и
системы связи

Программирование в
инфокоммуникационных сетях

Лабораторная работа #3

Коркина Анна Михайловна(К3120)

Преподаватель: Харьковская Татьяна Александровна

Г. Санкт-Петербург

22.07.2022

2 Задача. Компоненты

Условие:

Дан неориентированный граф с n вершинами и m ребрами. Нужно посчитать количество компонент связности в нем.

Решение:

```
c_nodes, c_ribs = map(int, inp.readline().split())

links = [[] for _ in range(c_nodes)]

for _ in range(c_ribs):
    node, link = map(int, inp.readline().split())
    links[node-1].append(link-1)
    links[link-1].append(node-1)

inp.close()

used = [False] * c_nodes
c_areas = 0

start = time.time()
for i in range(c_nodes):
    if not used[i]:
        c_areas += 1
        que = [i]
        used[i] = True
        while len(que) > 0:
            node = que.pop()
            for j in links[node]:
                if not used[j]:
                    que.append(j)
                    used[j] = True
```

В данном решении применяется алгоритм при помощи обычного поиска в глубину. Изначально поиск в глубину был реализован при помощи рекурсии, но было принято решение отказаться от рекурсии из-за возможности переполнения стека.

Тесты:

```
11 11
1 2
2 3
3 4
4 2
4 5
8 10
10 11
8 11
3 5
6 4
6 7
3
Time: 0.0
```

```
5
Time: 0.0 5 0|
```

Вывод:

Поиск в глубину лучше осуществлять с помощью очереди или стека.

1 Задача. Лабиринт

Условие:

Проверить, что существует путь между соответствующими двумя вершинами в графе

Решение:

```

c_nodes, c_ribs = map(int, inp.readline().split())

links = [[] for _ in range(c_nodes)]

for _ in range(c_ribs):
    node, link = map(int, inp.readline().split())
    links[node - 1].append(link - 1)
    links[link - 1].append(node - 1)

src, trg = map(int, inp.readline().split())

src -= 1
trg -= 1

inp.close()

used = [False] * c_nodes
que = [src]

exist = False
used[src] = True
while len(que) > 0:
    node = que.pop()
    if node == trg:
        exist = True
        break
    for j in links[node]:
        if not used[j]:
            que.append(j)
            used[j] = True

```

Решение так же основано на реализации поиска в глубину, только теперь из начальной вершины и проверки на наличие конечной вершины в этом компоненте связности.

Тесты:

```

11 11
1 2
2 3
3 4
4 2
4 5
8 10
10 11
8 11
3 5
6 4
6 7
7 1

```

1

```

11 11
1 2
2 3
3 4
4 2
4 5
8 10
10 11
8 11
3 5
6 4
6 7
8 4

```

0

Вывод:

Поиск в глубину может помочь при решении многих задач связанных с графами.

8 Задача. Стоимость полета

Условие:

Дан ориентированный граф с положительными весами ребер, n - количество вершин и m - количество ребер, а также даны две вершины u и v . Вычислить вес кратчайшего пути между u и v (то есть минимальный общий вес пути из u в v).

Решение:

```
c_nodes, c_ribs = map(int, inp.readline().split())
edges = []

for _ in range(c_ribs):
    node, link, weight = map(int, inp.readline().split())
    edges.append((node - 1, link - 1, weight))

src, dst = map(int, inp.readline().split())
src -= 1
dst -= 1

inp.close()

ways = [None] * c_nodes
ways[src] = 0

start = time.time()
for i in range(c_nodes - 1):
    for source, destination, weight in edges:
        if ways[source] is not None:
            ways[destination] = min((math.inf if (ways[destination] is None) else ways[destination]),
                                     ways[source] + weight)
end = time.time()

out = open("output.txt", "w")
print(f"Answer: {ways[dst]}, Time: {end - start}")
if ways[dst] is None:
    out.write("-1")
else:
    out.write(str(ways[dst]))
```



```

edges = []
ways = {}

for _ in range(c_ribs):
    node, trash, link = inp.readline().split()
    edges.append((node, link, 1))
    if (node not in ways.keys()):
        ways[node] = None
    if (link not in ways.keys()):
        ways[link] = None

src = inp.readline().removesuffix("\n")
dst = inp.readline().removesuffix("\n")

inp.close()

ways[src] = 0

for i in range(len(ways.keys())):
    for source, destination, weight in edges:
        if ways[source] is not None:
            ways[destination] = min((math.inf if (ways[destination] is None) else ways[destination]), ways[source] + weight)

```

В данной задаче я так же применила алгоритм Форда-Беллмана, только теперь место массива я хранила дневник, где ключ – элемент, который надо получить, а в качестве значения так же число. При считывании в качестве веса ребра устанавливается 1.

Тесты:

ID	Дата	Язык	Результат	Тест	Время	Память
17079273	02.05.2022 17:19:41	Python	Accepted		0,14	798 Кб
17079270	02.05.2022 17:18:34	Python	Accepted		0,125	794 Кб

12 Задача. Цветной лабиринт

Условие:

В выходной файл выведите строку INCORRECT, если описание пути некорректно, иначе выведите номер комнаты, в которую ведет описанный путь. Путь начинается в комнате номер один.

Решение:

```
c_moves = int(inp.readline())
moves = list(map(int, inp.readline().split()))

inp.close()

curr = 0
fail = False
for i in range(c_moves):
    move_color = moves[i] - 1
    if move_color not in maze[curr].keys():
        fail = True
        break
    new = maze[curr][move_color]
    if new is not None:
        curr = new
    else:
        fail = True
        break

out = open("output.txt", "w")
if fail:
    out.write("INCORRECT")
else:
    out.write(str(curr + 1))
out.close()
```


Решение основывается на том, чтобы у каждой комнаты хранить с какими цветами коридоров она соединена, и, если при проверке оказывается, что она не соединена с коридором нужного цвета цикл обрывается и устанавливается, что путь является некорректным. Если же цикл проходит полностью, то в переменной `curr` будет храниться комната, в которую должен прийти человек после пройденного пути.

Тесты:

Посылки решений:

ID	Дата	Язык	Результат	Тест	Время	Память
17089506	04.05.2022 12:17:51	Python	Accepted		0,14	11 Мб
17089499	04.05.2022 12:17:37	Python	Accepted		0,156	11 Мб

Вывод:

в этой задаче самое главное решить в каком виде хранить данные, так как от самого хранения зависит как быстро мы будем определять можно ли пройти из данной комнаты в другую.

14 Задача. Автобусы

Условие:Между некоторыми деревнями края Власюки ходят автобусы. Поскольку пассажиропотоки здесь не очень большие, то автобусы ходят всего несколько раз в день.

Марии Ивановне требуется добраться из деревни d в деревню v как можно быстрее (считается, что в момент времени 0 она находится в деревне d).

Решение:

```
c_villages = int(inp.readline())
start, end = map(int, inp.readline().split())
c_races = int(inp.readline())

buses = [[] for _ in range(c_races)]

for _ in range(c_races):
    src, src_time, dst, dst_time = map(int, inp.readline().split())
    buses[src - 1].append((src_time, dst - 1, dst_time))

times = [math.inf] * (c_villages)
times[start - 1] = 0

used = [False] * (c_villages)
while True:
    min_time = math.inf
    for i in range(c_villages):
        if not used[i] and times[i] < min_time:
            min_time = times[i]
            min_village = i
    if min_time == math.inf:
        break
    src = min_village
    used[src] = True
    for src_time, dst, dst_time in buses[src]:
        if times[src] <= src_time and dst_time < times[dst]:
            times[dst] = dst_time

out = open("output.txt", "w")
if times[end - 1] == math.inf:
    out.write("-1")
```

данном решении используется алгоритм Дейкстры, только немного модифицированный, место расстояний необходимо хранить минимальное время прибытие в деревню.

Тесты:

Посылки решений:

ID	Дата	Язык	Результат	Тест	Время	Память
17090603	04.05.2022 14:17:56	Python	Accepted		0,046	2538 Kb

Вывод:

Данная задача сводится к изменению взгляда на алгоритм, путём замены расстояния на время прибытия.

4 Задача. Порядок курсов

Дан ориентированный ациклический граф (DAG) с n вершинами и m ребрами. Выполните топологическую сортировку.

Решение:

Пишем простой обход в глубину, и для каждой вершины после обхода ее связей добавляем ее в начало стека: чем раньше вышли из цикла вершины – тем меньше у нее должна быть позиция.

```
def explore(v, visited, stack):
    visited[v] = True
    for i in graph[v]:
        if not visited[i]:
            explore(i, visited, stack)
    stack.insert(0, v)

def dfs(G):
    visited = [False] * n
    stack = []
    for i in range(n):
        if not visited[i]:
            explore(i, visited, stack)
    return stack
```

Выводы:

Данная лабораторная работа демонстрирует, что зачастую можно использовать одни и те же алгоритмы, с учётом предварительно преобразования исходных данных для корректной работы алгоритмов.