Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
ПО КУРСУ «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»
ТЕМА: ГРАФЫ
ВАРИАНТ 10

Выполнил:
Соболев А. А
K3240

Проверила:
Ромакина О. М.

Санкт-Петербург
2025

СОДЕРЖАНИЕ

ЗАДАЧИ ПО ВАРИАНТУ

5 задача. Город с односторонним движением (1.5 балла)

Постановка задачи

Нужно вычислить количество компонентов сильной связности заданного ориентированного графа с n вершинами и m ребрами.

Листинг кода

```
import time, tracemalloc

def dfs1(node, graph, visited, stack):
    visited[node] = True
    for neighbor in graph[node]:
        if not visited[neighbor]:
            dfs1(neighbor, graph, visited, stack)
    stack.append(node)

def dfs2(node, transposed_graph, visited):
    visited[node] = True
    for neighbor in transposed_graph[node]:
        if not visited[neighbor]:
            dfs2(neighbor, transposed_graph, visited)

def kosaraju_scc(n, edges):
    graph = {i: [] for i in range(1, n + 1)}
    transposed_graph = {i: [] for i in range(1, n + 1)}
    visited = {i: False for i in range(1, n + 1)}
    stack = []

    for u, v in edges:
        graph[u].append(v)
        transposed_graph[v].append(u)

    for node in range(1, n + 1):
        if not visited[node]:
            dfs1(node, graph, visited, stack)

    visited = {i: False for i in range(1, n + 1)}
    scc_count = 0
```

```
    while stack:
        node = stack.pop()
        if not visited[node]:
            dfs2(node, transposed_graph, visited)
            scc_count += 1

    return scc_count

with open("input.txt", "r") as f:
    n, m = map(int, f.readline().split())
    edges = [tuple(map(int, f.readline().split())) for _ in range(m)]


start_time = time.perf_counter()
tracemalloc.start()

result = kosaraju_scc(n, edges)

end_time = time.perf_counter()

current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

with open("output.txt", "w") as f:
    f.write(str(result) + "\n")
```

Объяснение

Я использовал алгоритм Косарайо. Я создал словари, в одном из которых хранится обычный граф, а в другом транспортированный. Для каждой непосещенной вершины я запускаю обход в глубину и добавляю вершины в стек. Затем я из стека по очереди достаю вершины и обхожу их с транспортированным графом. В транспонированном графе всё ребра инвертируются и это означает, что в если в обычном графе можно дойти из вершины 1 в вершину 2, в транспортированном можно дойти из вершины 2 в вершину 1. Транспонированный граф позволяет собрать всю компоненту

сильной связности целиком. Если бы граф не транспонировали, то компоненты могли бы пересекаться.

Время и память

- Время выполнения: 0.000889 секунд (5 секунд ограничение)
- Использовано памяти: 0.001068 Мб (512 мб ограничение)

Примеры

Вводный файл	Выходной файл
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

Вывод

В данной задаче я научился вычислять компоненты сильной связности ориентированного графа.

8 задача. Стоимость полета (1.5 балла)

Постановка задачи

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).

Листинг кода

```
import heapq, time, tracemalloc
def dijkstra(n, edges, start, end):
    graph = {i: [] for i in range(1, n + 1)}
    for u, v, w in edges:
        graph[u].append((v, w))
    min_dist = {i: float('inf') for i in range(1, n + 1)}
    min_dist[start] = 0
    pq = [(0, start)]
    while pq:
        cur_dist, cur_node = heapq.heappop(pq)
        if cur_dist > min_dist[cur_node]:
            continue
        for neighbor, weight in graph[cur_node]:
            new_dist = cur_dist + weight
            if new_dist < min_dist[neighbor]:
                min_dist[neighbor] = new_dist
                heapq.heappush(pq, (new_dist, neighbor))
    return min_dist[end] if min_dist[end] != float('inf') else -1
with open("input.txt", "r") as f:
    n, m = map(int, f.readline().split())
    edges = [tuple(map(int, f.readline().split())) for _ in range(m)]
    u, v = map(int, f.readline().split())
start_time = time.perf_counter()
tracemalloc.start()
result = dijkstra(n, edges, u, v)
end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")
```

```
with open("output.txt", "w") as f:
    f.write(str(result) + "\n")
```

Объяснение

Я использовал алгоритм Дейкстры, он позволяет найти минимальную стоимость в графе. Я создал словарь с графом и использовал кучу. Кучу я выбрал, потому что она хранит всегда первым минимальный элемент, что нужно для задачи. Пока в куче были вершины, я извлекал оттуда элемент и сравнивал расстояние. Если новый вес был меньше, то я перезаписывал и клал элемент в кучу.

Время и память

- Время выполнения: 0.000061 секунд (10 секунд ограничение)
- Использовано памяти: 0.000793 Мб (512 мб ограничение)

Примеры

Вводный файл	Выходной файл
3 3 1 2 7 1 3 5 2 3 2 3 2	-1
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

Вывод

В данной задаче я научился применять алгоритм Дейкстры для поиска кратчайшего пути в ориентированном взвешенном графе

11 задача. Алхимия (3 балла)

Постановка задачи

Задан набор алхимических реакций, описанных на найденных глиняных табличках, исходное вещество и требуемое вещество. Необходимо выяснить: возможно ли преобразовать исходное вещество в требуемое с помощью этого набора реакций, а в случае положительного ответа на этот вопрос - найти минимальное количество реакций, необходимое для осуществления такого преобразования.

Листинг кода

```
from collections import deque
import tracemalloc, time

def bfs(reactions, start, target):
    if start == target:
        return 0

    graph = {}
    for reaction in reactions:
        src, dest = reaction.split(" -> ")
        if src not in graph:
            graph[src] = []
        graph[src].append(dest)

    queue = deque([(start, 0)])
    visited = {start}

    while queue:
        current, steps = queue.popleft()

        if current not in graph:
            continue

        for neighbor in graph[current]:
            if neighbor == target:
                return steps + 1
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append((neighbor, steps + 1))
```

```
    return -1


with open("input.txt", "r") as f:
    m = int(f.readline().strip())
    reactions = [f.readline().strip() for _ in range(m)]
    start = f.readline().strip()
    target = f.readline().strip()

start_time = time.perf_counter()
tracemalloc.start()

result = bfs(reactions, start, target)

end_time = time.perf_counter()

current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

with open("output.txt", "w") as f:
    f.write(str(result) + "\n")
```

Объяснение задачи

Я составил граф (от каждого вещества - списки веществ, в которые оно превращается), а затем с помощью обхода в ширину (BFS) нашёл минимальное число реакций от начального вещества до целевого. Если путь не найден, возвращаю -1

Время и память

- Время выполнения: 0.000075 секунд (1 секунда ограничение)
- Использовано памяти: 0.001626 Мб (16 Мб ограничение)

Примеры

Вводный файл	Выходной файл
5 Aqua -> AquaVita AquaVita -> PhilosopherStone AquaVita -> Argentum Argentum -> Aurum AquaVita -> Aurum Aqua Aurum	2
5 Aqua -> AquaVita AquaVita -> PhilosopherStone AquaVita -> Argentum Argentum -> Aurum AquaVita -> Aurum Aqua Osmium	2

Проверка на acmp

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
23054724	13.03.2025 1:23:42	Соболев Артём Анатольевич	0743	PyPy	Accepted		0,265	4980 Кб

Вывод

В данном задаче я применил алгоритм поиска в ширину для нахождения кратчайшего пути реакций

ДОПОЛНИТЕЛЬНЫЕ ЗАДАЧИ

1 задача. Лабиринт (1 балл)

Постановка задачи

Вам дан неориентированный граф и две различные вершины u и v. Проверьте, есть ли путь между u и v.

Листинг кода

```
import time, tracemalloc

def dfs(graph, start, visited):
    stack = [start]
    while stack:
        node = stack.pop()
        if node in visited:
            continue
        visited.add(node)
        stack.extend(graph[node])


def is_path_exists(n, edges, u, v):
    graph = {i: [] for i in range(1, n + 1)}
    for a, b in edges:
        graph[a].append(b)
        graph[b].append(a)

    visited = set()
    dfs(graph, u, visited)
    return 1 if v in visited else 0

with open("input.txt", "r") as f:
    n, m = map(int, f.readline().split())
    edges = [tuple(map(int, f.readline().split())) for _ in range(m)]
    u, v = map(int, f.readline().split())

start_time = time.perf_counter()
tracemalloc.start()

result = is_path_exists(n, edges, u, v)
```

```
end_time = time.perf_counter()

current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

with open("output.txt", "w") as f:
    f.write(str(result) + "\n")
```

Объяснение

Для решения задачи я использовал поиск в глубину (dfs). Для начала я создаю список смежности - это словарь, где ключ - это вершина, а значение - список соседей. Я использую список как стек, в котором изначально лежит вершина, из которой нужно идти. Я убираю эту вершину из стека, добавляю в множество посещенных вершин, в стек кладу всех соседей этой вершины и повторяю цикл до тех пор, пока в стеке есть вершины. Если в множестве есть конечная вершина, которую нужно посетить, то значит до нее можно дойти из начальной вершины

Время и память

- Время выполнения: 0.001158 секунд (5 секунд ограничение)
- Использовано памяти: 0.000732 Мб (512 мб ограничение)

Примеры

Вводный файл	Выходной файл
4 2 1 2 3 2 1 4	0
4 4 1 2 3 2 4 3 1 4 1 4	1

Вывод

В данной задаче я научился использовать поиск в глубину для проверки существования пути между двумя вершинами неориентированного графа.

2 задача. Компоненты (1 балл)

Постановка задачи

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

Листинг кода

```
import time, tracemalloc
def dfs(graph, node, visited):
    stack = [node]
    while stack:
        v = stack.pop()
        if v not in visited:
            visited.add(v)
            stack.extend(graph[v])
def count_connected_components(n, edges):
    graph = {i: [] for i in range(1, n + 1)}
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)
    visited = set()
    components = 0
    for node in range(1, n + 1):
        if node not in visited:
            components += 1
            dfs(graph, node, visited)
    return components
with open("input.txt", "r") as f:
    n, m = map(int, f.readline().split())
    edges = [tuple(map(int, f.readline().split())) for _ in range(m)]
start_time = time.perf_counter()
tracemalloc.start()
result = count_connected_components(n, edges)
end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")
with open("output.txt", "w") as f:
    f.write(str(result) + "\n")
```

Объяснение

В этой задаче я использовал алгоритм dfs. Я создал граф и добавил в него вершины и куда можно попасть из вершины. Затем я создал пустое множество, в которое буду записывать посещенные вершины. Если вершина не посещена, то я запускаю алгоритм dfs и она добавляет все посещенные вершины. Счетчик компонент связности увеличивается на 1 раз каждый раз, когда находится непосещенная вершина в главной функции.

Время и память

- Время выполнения: 0.000022 секунд (5 секунд ограничение)
- Использовано памяти: 0.000526 Мб (512 мб ограничение)

Примеры

Вводный файл	Выходной файл
4 2 1 2 3 2	2

Вывод

В данной задаче я научился находить и подсчитывать компоненты связности в неориентированном графе с помощью алгоритма dfs.

3 задача. Циклы (1 балл)

Постановка задачи

Проверьте, содержит ли данный граф циклы.

Листинг кода

```
import time, tracemalloc
def dfs(node, graph, visited, stack):
    visited[node] = "GRAY"
    for neighbor in graph[node]:
        if visited[neighbor] == "GRAY":
            return True
        if visited[neighbor] == "WHITE":
            if dfs(neighbor, graph, visited, stack):
                return True
    visited[node] = "BLACK"
    return False
def has_cycle(n, edges):
    graph = {i: [] for i in range(1, n + 1)}
    for u, v in edges:
        graph[u].append(v)
    visited = {i: "WHITE" for i in range(1, n + 1)}
    for node in range(1, n + 1):
        if visited[node] == "WHITE":
            if dfs(node, graph, visited, []):
                return 1
    return 0
with open("input.txt", "r") as f:
    n, m = map(int, f.readline().split())
    edges = [tuple(map(int, f.readline().split())) for _ in range(m)]
start_time = time.perf_counter()
tracemalloc.start()
result = has_cycle(n, edges)
end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

with open("output.txt", "w") as f:
    f.write(str(result) + "\n")
```

Объяснение

Я создал словарь graph, где для каждой вершины хранится список её соседей, и словарь visited, в котором каждой вершине присваиваю цвет WHITE. Функция dfs при входе в вершину помечает её как GRAY, затем для каждого соседа проверяет: если сосед уже серый, возвращает True, если белый - запускает dfs рекурсивно. После обхода всех соседей вершина помечается как BLACK и, если цикла не было, возвращается False. В has_cycle инициализирую graph и visited, пробегаю по всем вершинам и, если dfs для какой-то белой вершины вернул True, сразу возвращаю 1, иначе после всех запусков возвращаю 0.

Время и память

- Время выполнения: 0.000021 секунд (5 секунд ограничение)
- Использовано памяти: 0.000595 Мб

Примеры

Вводный файл	Выходной файл
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

Вывод

В данной задаче я научился строить список смежности и применять DFS с трёхцветной раскраской вершин для обнаружения циклов в ориентированном графе.

4 задача. Порядок курсов (1 балл)

Постановка задачи

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

Листинг кода

```
from collections import deque
import time, tracemalloc

def topological_sort(n, edges):
    graph = {i: [] for i in range(1, n + 1)}
    in_degree = {i: 0 for i in range(1, n + 1)}

    for u, v in edges:
        graph[u].append(v)
        in_degree[v] += 1

    queue = deque([node for node in range(1, n + 1) if in_degree[node] ==
0])
    topo_order = []

    while queue:
        node = queue.popleft()
        topo_order.append(node)

        for neighbor in graph[node]:
            in_degree[neighbor] -= 1
            if in_degree[neighbor] == 0:
                queue.append(neighbor)

    return topo_order

with open("input.txt", "r") as f:
    n, m = map(int, f.readline().split())
    edges = [tuple(map(int, f.readline().split())) for _ in range(m)]

start_time = time.perf_counter()
tracemalloc.start()

result = topological_sort(n, edges)
```

```
end_time = time.perf_counter()

current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

with open("output.txt", "w") as f:
    f.write(" ".join(map(str, result)) + "\n")
```

Объяснение

Я создал словарь графа и словарь со степенями вершины. В словаре со степенями вершины я увеличиваю значение, если в вершину ведет другая вершина. После чего я в очередь добавляю вершины, в которые не ведут другие вершины. Они могут быть первыми в топологическом порядке. Из очереди я доставал первые вершины, которые были туда добавлены и добавляю в топологическую сортировку. Затем я уменьшаю степень вершины каждому соседу этой вершины и добавляю в очередь эту вершину

Время и память

- Время выполнения: 0.000050 секунд (10 секунд ограничение)
- Использовано памяти: 0.001198 Мб (512 мб ограничение)

Примеры

Входной файл	Выходной файл
4 1 3 1	2 3 4 1
4 3 1 2 4 1 3 1	3 4 1 2
5 7 2 1 3 2 3 1 4 3 4 1 5 2 5 3	4 5 3 2 1

Вывод

В данной задаче я научился выполнять топологическую сортировку ориентированного ациклического графа.

6 задача. Количество пересадок (1 балл)

Поставка задачи

Дан неориентированный граф с n вершинами и m ребрами, а также две вершины u и v, нужно посчитать длину кратчайшего пути между u и v (то есть, минимальное количество ребер в пути из u в v).

Листинг кода

```
from collections import deque
import time, tracemalloc

def bfs_shortest_path(n, edges, start, end):
    graph = {i: [] for i in range(1, n + 1)}
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)

    queue = deque([(start, 0)])
    visited = {start}

    while queue:
        node, dist = queue.popleft()
        if node == end:
            return dist
        for neighbor in graph[node]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append((neighbor, dist + 1))

    return -1

with open("input.txt", "r") as f:
    n, m = map(int, f.readline().split())
    edges = [tuple(map(int, f.readline().split())) for _ in range(m)]
    u, v = map(int, f.readline().split())

start_time = time.perf_counter()
tracemalloc.start()

result = bfs_shortest_path(n, edges, u, v)
```

```
end_time = time.perf_counter()

current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")



with open("output.txt", "w") as f:
    f.write(str(result) + "\n")
```

Объяснение

Я использовал алгоритм в ширину. Я создал словарь с графом и поместил в очередь первую вершину. Я взял очередь, потому что она позволяет извлекать вершину самой первой помещенной (принцип FIFO). Я достаю из вершины самую первую посещенную вершину и добавляю в очередь его соседей, увеличивая дистанцию. Когда найдется вершина, то расстояние будет кратчайшим благодаря использовании очереди.

Время и память

- Время выполнения: 0.000075 секунд (10 секунд ограничение)
- Использовано памяти: 0.001282 Мб (512 мб ограничение)

Примеры

Входной файл	Выходной файл
5 4 5 2 1 3 3 4 1 4 3 5	-1
4 4 1 2 4 1 2 3 3 1 2 4	2

Вывод

В данной задаче я научился находить минимальное число ребер в пути между двумя вершинами неориентированного графа, применяя поиск в ширину.

13 задача. Грядки (3 балла)

Постановка задачи

Прямоугольный садовый участок шириной N и длиной M метров разбит на квадраты со стороной 1 метр. На этом участке вскопаны грядки. Грядкой называется совокупность квадратов, удовлетворяющая таким условиям:

- из любого квадрата этой грядки можно попасть в любой другой квадрат этой же грядки, последовательно переходя по грядке из квадрата в квадрат через их общую сторону;
- никакие две грядки не пересекаются и не касаются друг друга ни по вертикальной, ни по горизонтальной сторонам квадратов (касание грядок углами квадратов допускается).

Подсчитайте количество грядок на садовом участке.

Листинг кода

```
import time, tracemalloc

def is_valid(x, y):
    return 0 <= x < N and 0 <= y < M and garden[x][y] == '#'

def dfs(x, y):
    stack = [(x, y)]
    garden[x][y] = '.'
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    while stack:
        cx, cy = stack.pop()
        for dx, dy in directions:
            nx, ny = cx + dx, cy + dy
            if is_valid(nx, ny):
                garden[nx][ny] = '.'
                stack.append((nx, ny))

start_time = time.perf_counter()
tracemalloc.start()

with open("input.txt", "r") as f:
    N, M = map(int, f.readline().split())
    garden = [list(f.readline().strip()) for _ in range(N)]
```

```
count = 0
for i in range(N):
    for j in range(M):
        if garden[i][j] == '#':
            dfs(i, j)
            count += 1

end_time = time.perf_counter()

current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

with open("output.txt", "w") as f:
    f.write(str(count) + "\n")
```

Объяснение кода

Я использовал алгоритм dfs. Как только нашлось #, начинаем расширяться по её соседям в доступных направлениях, пока не закончится грядка. Все посещенные клетки превращаются в точки, чтобы их больше не учитывать. После этого увеличиваем счетчик на один. Так перебирается вся матрица со значениями

Время и память

- Время выполнения: 0.000219 секунд (1 секунда ограничение)
- Использовано памяти: 0.018090 Мб (16 мб ограничение)

Примеры

Вводный файл	Выходной файл
5 10 ##.....#. .#..#...#. .###....#. ..##....#. ....#.	3
5 10 ##..#####. .#.#.#.... ###..##.#. ..##.....# .###.#####	5

Проверка на acmp

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
23279517	02.05.2025 20:43:13	Соболев Артём Анатольевич	0432	PyPy	Accepted		0,25	3352 Кб

Вывод

В данной задаче я научился находить количество связных областей в матрице.

14 задача. Автобусы (3 балла)

Постановка задачи

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

Листинг кода

```
import heapq, time, tracemalloc

def dijkstra(n, d, v, bus_routes):
    graph = {i: [] for i in range(1, n + 1)}

    for u, start_time, dest, arrival_time in bus_routes:
        graph[u].append((start_time, dest, arrival_time))

    min_time = {i: float('inf') for i in range(1, n + 1)}
    min_time[d] = 0

    pq = [(0, d)]

    while pq:
        cur_time, cur_village = heapq.heappop(pq)

        if cur_village == v:
            return cur_time

        for start_time, next_village, arrival_time in graph[cur_village]:
            if cur_time <= start_time and arrival_time <
min_time[next_village]:
                min_time[next_village] = arrival_time
                heapq.heappush(pq, (arrival_time, next_village))

    return -1

with open("input.txt", "r") as f:
    n = int(f.readline().strip())
    d, v = map(int, f.readline().split())
    r = int(f.readline().strip())
    bus_routes = [tuple(map(int, f.readline().split())) for _ in range(r)]
```

```
start_time = time.perf_counter()
tracemalloc.start()

result = dijkstra(n, d, v, bus_routes)

end_time = time.perf_counter()

current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_memory / (1024 * 1024)):.6f} Мб")

with open("output.txt", "w") as f:
    f.write(str(result) + "\n")
```

Объяснение кода

Я использовал алгоритм Дейкстры для решения задачи. Создал словарь, в котором ключ - это деревня, а значение - кортеж из времени начала поездки, дистанции и времени прибытия в эту деревню, затем его запомнил. После я взял кучу, так как с ее помощью извлечение кучи происходит быстрее. Затем я запустил цикл по куче и сравнивал время с минимальным.

Время и память

- Время выполнения: 0.000028 секунд (1 секунда ограничение)
- Использовано памяти: 0.000793 Мб (16 Мб ограничение)

Примеры

Вводный файл	Выходной файл
3 1 3 4 1 0 2 5 1 1 2 3 2 3 3 5 1 1 3 10	5

Проверка на acmp

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
23054691	13.03.2025 0:44:07	Соболев Артём Анатольевич	0134	PyPy	Accepted		0,265	3016 Кб

Вывод

В данной задаче я применил алгоритм Дейкстры для поиска минимального пути.

15 задача. Герои (3 балла)

Постановка задачи

Коварный кардинал Ришелье вновь организовал похищение подвесок королевы Анны; вновь спасать королеву приходится героическим мушкетерам. Атос, Портос, Арамис и д’Артаньян уже перехватили агентов кардинала и вернули украденное; осталось лишь передать подвески королеве Анне. Королева ждет мушкетеров в дворцовом саду. Дворцовый сад имеет форму прямоугольника и разбит на участки, представляющие собой небольшие садики, содержащие коллекции растений из разных климатических зон. К сожалению, на некоторых участках, в том числе на всех участках, расположенных на границах сада, уже притаились в засаде гвардейцы кардинала; на бой с ними времени у мушкетеров нет. Мушкетерам удалось добыть карту сада с отмеченными местами засад; теперь им предстоит выбрать оптимальные пути к королеве. Для надежности друзья разделили между собой спасенные подвески и проникли в сад поодиночке, поэтому начинают свой путь к королеве с разных участков сада. Двигаются герои по максимально короткой возможной траектории. Марлезонский балет вот-вот начнется; королева не в состоянии ждать героев больше L минут; ровно в начале L+1-й минуты королева покинет парк, и те мушкетеры, что не успеют к этому времени до нее добраться, не смогут передать ей подвески. На преодоление одного участка у мушкетеров уйдет ровно по минуте. С каждого участка мушкетеры могут перейти на 4 соседние.

Требуется выяснить, сколько подвесок будет красоваться на платье королевы, когда она придет на бал.

Листинг кода

```
import time
import tracemalloc
from collections import deque

tracemalloc.start()
start_time = time.perf_counter()

with open('input.txt') as f:
    n, m = map(int, f.readline().split())
    g = [f.readline().strip() for _ in range(n)]
    qx, qy, L = map(int, f.readline().split())
    starts = [tuple(map(int, f.readline().split())) for _ in range(4)]

dist = [[-1] * m for _ in range(n)]
dq = deque()
dq.append((qx-1, qy-1))
dist[qx-1][qy-1] = 0

while dq:
    x, y = dq.popleft()
    for dx, dy in ((1,0),(-1,0),(0,1),(0,-1)):
        nx, ny = x + dx, y + dy
        if 0 <= nx < n and 0 <= ny < m and g[nx][ny] == '0' and dist[nx][ny]
== -1:
            dist[nx][ny] = dist[x][y] + 1
            dq.append((nx, ny))

res = 0
for x, y, p in starts:
    d = dist[x-1][y-1]
    if d != -1 and d <= L:
        res += p
with open('output.txt', 'w') as f:
    f.write(str(res))
end_time = time.perf_counter()
current_mem, peak_mem = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Время выполнения: {(end_time - start_time):.6f} секунд")
print(f"Использовано памяти: {(peak_mem / (1024 * 1024)):.6f} Мб")
```

Объяснение кода

Я использовал алгоритм поиска в ширину для решения задачи. Создал двумерный массив dist, где в ячейке с координатами хранится минимальное число шагов (минут) от королевы до этой клетки. Стартовую точку инициализировал нулём, остальные - как не посещённые (-1). Затем взял очередь deque, так как она позволяет очень быстро добавлять и извлекать элементы по принципу FIFO. Запустил цикл по этой очереди. Для каждой извлечённой клетки перебирал четыре соседние - если сосед внутри границ, свободен и ещё не посещён, записывал в dist расстояние текущей + 1 и добавлял его в очередь. Для каждого мушкетёра проверил: если до его стартовой точки можно добраться за не более L шагов, прибавил к ответу число подвесок у него.

Время и память

- Время выполнения: 0.000877 секунд
- Использовано памяти: 0.016982 Мб (16 Мб ограничение)

Примеры

Входной файл	Выходной файл
5 5 11111 10001 10001 10001 11111 4 4 10 2 2 1 2 3 2 3 2 3 3 3 4	10

5 5 11111 10001 10111 10101 11111 4 4 10 2 2 1 2 2 2 2 2 3 2 2 4	0

Проверка на acmp

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
23298556	08.05.2025 12:38:40	Соболев Артём Анатольевич	0846	PyPy	Accepted		0.265	4972 Кб

Вывод

В данной задаче я научился выполнять многократный алгоритм поиска в ширину из разных стартовых точек.

ВЫВОД

В данном лабораторной работе я изучал графы и алгоритмы для них: поиск в глубину и ширину для проверки достижимости и кратчайших путей в невзвешенных графах, алгоритм Косарайю, алгоритм Дейкстры, топологическую сортировку.