Лабораторная работа 3

090304-РПИб-о24

Чучкалов Александр

**Задание:**

Матрица размера N\*M определяет некоторый лабиринт. B матрице элемент 1 обозначает стену, а 0 определяет свободное место. В первой строке матрицы определяются входы x(i), а в последней выходы y(i), i=1,..,k, которые должны быть нулевыми элементами.

Необходимо определить, можно ли

а) провести k человек от входа x(i) до выхода y(i) соответственно, i=1,..,k, таким образом, чтобы каждое свободное место посещалось не более одного раза.

б) то же, но человека можно выводить чеpез любой из выходов. Примечание: Движение в лабиринте осуществляется только по вертикали или горизонтали.

**Код:**

import time

import random

from collections import deque

from queue import LifoQueue

class MazeSolver:

def \_\_init\_\_(self, maze):

self.maze = maze

self.n = len(maze)

self.m = len(maze[0]) if self.n > 0 else 0

self.visited = [[False for \_ in range(self.m)] for \_ in range(self.n)]

def reset\_visited(self):

self.visited = [[False for \_ in range(self.m)] for \_ in range(self.n)]

def get\_entrances(self):

entrances = []

if self.n == 0:

return entrances

for j in range(self.m):

if self.maze[0][j] == 0:

entrances.append((0, j))

return entrances

def get\_exits(self):

exits = []

if self.n == 0:

return exits

for j in range(self.m):

if self.maze[self.n - 1][j] == 0:

exits.append((self.n - 1, j))

return exits

def is\_valid(self, x, y):

return 0 <= x < self.n and 0 <= y < self.m and self.maze[x][y] == 0 and not self.visited[x][y]

# Реализация DFS с использованием стека на массиве

def find\_path\_array\_stack(self, start, end):

if not self.is\_valid(start[0], start[1]) or not self.is\_valid(end[0], end[1]):

return False

self.reset\_visited()

stack = []

stack.append((start[0], start[1], []))

while stack:

x, y, path = stack.pop()

if (x, y) == end:

return True

if self.visited[x][y]:

continue

self.visited[x][y] = True

# Добавляем соседей в стек

for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

nx, ny = x + dx, y + dy

if self.is\_valid(nx, ny):

stack.append((nx, ny, path + [(x, y)]))

return False

# Реализация DFS с использованием стека на связанном списке

def find\_path\_linked\_list\_stack(self, start, end):

if not self.is\_valid(start[0], start[1]) or not self.is\_valid(end[0], end[1]):

return False

self.reset\_visited()

stack = deque()

stack.append((start[0], start[1], []))

while stack:

x, y, path = stack.pop()

if (x, y) == end:

return True

if self.visited[x][y]:

continue

self.visited[x][y] = True

# Добавляем соседей в стек

for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

nx, ny = x + dx, y + dy

if self.is\_valid(nx, ny):

stack.append((nx, ny, path + [(x, y)]))

return False

# Реализация DFS с использованием стека из стандартной библиотеки

def find\_path\_std\_stack(self, start, end):

if not self.is\_valid(start[0], start[1]) or not self.is\_valid(end[0], end[1]):

return False

self.reset\_visited()

stack = LifoQueue()

stack.put((start[0], start[1], []))

while not stack.empty():

x, y, path = stack.get()

if (x, y) == end:

return True

if self.visited[x][y]:

continue

self.visited[x][y] = True

# Добавляем соседей в стек

for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

nx, ny = x + dx, y + dy

if self.is\_valid(nx, ny):

stack.put((nx, ny, path + [(x, y)]))

return False

# Решение задачи (a) - каждый человек идет к своему выходу

def solve\_a(self, stack\_type='array'):

entrances = self.get\_entrances()

exits = self.get\_exits()

k = min(len(entrances), len(exits))

if k == 0:

return False

# Для каждого входа ищем путь к соответствующему выходу

for i in range(k):

start = entrances[i]

end = exits[i]

if stack\_type == 'array':

found = self.find\_path\_array\_stack(start, end)

elif stack\_type == 'linked\_list':

found = self.find\_path\_linked\_list\_stack(start, end)

else:

found = self.find\_path\_std\_stack(start, end)

if not found:

return False

return True

# Решение задачи (b) - люди могут выходить через любой выход

def solve\_b(self, stack\_type='array'):

entrances = self.get\_entrances()

exits = self.get\_exits()

k = len(entrances)

if k == 0 or len(exits) == 0:

return False

# Для каждого входа ищем путь к любому выходу

for i in range(k):

start = entrances[i]

found = False

for end in exits:

self.reset\_visited()

if stack\_type == 'array':

path\_found = self.find\_path\_array\_stack(start, end)

elif stack\_type == 'linked\_list':

path\_found = self.find\_path\_linked\_list\_stack(start, end)

else:

path\_found = self.find\_path\_std\_stack(start, end)

if path\_found:

found = True

break

if not found:

return False

return True

def generate\_random\_maze(n, m, wall\_prob=0.3):

maze = [[1 if random.random() < wall\_prob else 0 for \_ in range(m)] for \_ in range(n)]

# Гарантируем хотя бы один вход и выход

for j in range(m):

if maze[0][j] == 1 and random.random() < 0.5:

maze[0][j] = 0

if maze[n - 1][j] == 1 and random.random() < 0.5:

maze[n - 1][j] = 0

return maze

def test\_performance():

# Генерируем один случайный размер лабиринта от 1 до 100

n, m = random.randint(1, 100), random.randint(1, 100)

print(f"Тестируем лабиринт размером {n}x{m}")

# Генерируем лабиринт

maze = generate\_random\_maze(n, m)

solver = MazeSolver(maze)

# Проверяем количество входов и выходов

entrances = solver.get\_entrances()

exits = solver.get\_exits()

print(f"Найдено входов: {len(entrances)}, выходов: {len(exits)}")

if not entrances or not exits:

print("Не найдено входов или выходов, тестирование невозможно")

return

# Сравниваем производительность для каждой реализации стека

stack\_types = ['array', 'linked\_list', 'std']

results = {'a': {}, 'b': {}}

for stack\_type in stack\_types:

print(f"\nТестируем реализацию стека: {stack\_type}")

# Тестируем задачу (a)

start\_time = time.time()

result\_a = solver.solve\_a(stack\_type)

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

results['a'][stack\_type] = (result\_a, time\_a)

print(f"Задача (а) - свой выход для каждого: результат={'Да' if result\_a else 'Нет'}, время={time\_a:.6f}с")

# Тестируем задачу (b)

start\_time = time.time()

result\_b = solver.solve\_b(stack\_type)

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

results['b'][stack\_type] = (result\_b, time\_b)

print(f"Задача (б) - любой выход подходит: результат={'Да' if result\_b else 'Нет'}, время={time\_b:.6f}с")

# Выводим сравнительные результаты

print("\nСравнительные результаты:")

print("\nЗадача (а) - каждый человек к своему выходу:")

for stack\_type in stack\_types:

result, t = results['a'][stack\_type]

print(f"{stack\_type:12s}: {'Да' if result else 'Нет'}, время = {t:.6f}с")

print("\nЗадача (б) - можно использовать любой выход:")

for stack\_type in stack\_types:

result, t = results['b'][stack\_type]

print(f"{stack\_type:12s}: {'Да' if result else 'Нет'}, время = {t:.6f}с")

if \_\_name\_\_ == "\_\_main\_\_":

random.seed(42) # Для воспроизводимости результатов

test\_performance()

**Реузльтат:**

Работу выполнил Чучкалов Александр 090304-РПИб-о24

------------------------

Тестируем лабиринт размером 82x15

Найдено входов: 11, выходов: 14

Тестируем реализацию стека: array

Задача (а) - свой выход для каждого: результат=Нет, время=0.009278с

Задача (б) - любой выход подходит: результат=Нет, время=0.058225с

Тестируем реализацию стека: linked\_list

Задача (а) - свой выход для каждого: результат=Нет, время=0.003744с

Задача (б) - любой выход подходит: результат=Нет, время=0.053895с

Тестируем реализацию стека: std

Задача (а) - свой выход для каждого: результат=Нет, время=0.008996с

Задача (б) - любой выход подходит: результат=Нет, время=0.125664с

Сравнительные результаты:

Задача (а) - каждый человек к своему выходу:

array : Нет, время = 0.009278с

linked\_list : Нет, время = 0.003744с

std : Нет, время = 0.008996с

Задача (б) - можно использовать любой выход:

array : Нет, время = 0.058225с

linked\_list : Нет, время = 0.053895с

std : Нет, время = 0.125664с