Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

**Лабораторная работа №5**

**Вариант №1**

Выполнил:

Бацанова Е. А.

Проверил

Мусаев А.А.

Санкт-Петербург,

2024

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_fmjc2tazxxvi)

[Задание 1 4](#_ffln2tea40r9)

[Задание 2 6](#_f1i7y2ewctsl)

[Задание 3 10](#_9dpw69av3up2)

[ЗАКЛЮЧЕНИЕ 13](#_3h5j4q69usjz)

[СПИСОК ЛИТЕРАТУРЫ 14](#_ft5c8dlk3ymw)

[ПРИЛОЖЕНИЕ 15](#_1pjf9s5jjz9l)

# **ВВЕДЕНИЕ**

Целью данной работы является изучение алгоритмов поиска в графах и их применение для решения различных задач. В рамках работы мы будем рассматривать задачи, связанные с:

1. определением правильности скобочной структуры;
2. решением задач с помощью поиска в глубину и ширину;
3. поиском выхода из лабиринта на матрице, заполненной нулями и единицами.

# **Задание 1**

**Задание:** Написать программу, которая определяет, является ли введенная скобочная структура правильной.

Примеры правильных скобочных выражений: (), (())(), ()(), ((()))

Примеры скобочных выражений: )(, ())((), (, )))), ((().

Найдите порядковый номер первого символа (скобки), нарушающего правильность расстановки скобок.

**Решение:**

Проверка правильности последовательности скобочных выражений производится посредством функции *brackets(s)*. Внутри функции создается переменная *cnt*, которая будет отслеживать “баланс” скобок. Происходит проход по всем символам в строке *s*. Если символ ')' встречается, то *cnt* уменьшается, в противном случае увеличивается. Если на каком-то момент *cnt* становится отрицательным, то функция возвращает *False* и номер элемента, на котором нарушено условие. Если после обхода всех символов *cnt* равен 0, то возвращается *True* (то есть количество открывающихся и закрывающихся скобок равно и при этом ни разу закрывающаяся скобка не появилась раньше соответствующей ей открывающейся), иначе *False* (то есть количество открывающихся и закрывающихся скобок не равно).

Проверим работу программы: в бесконечном цикле будем запрашивать пользовательский ввод последовательности скобок и вызывать функцию *brackets*.

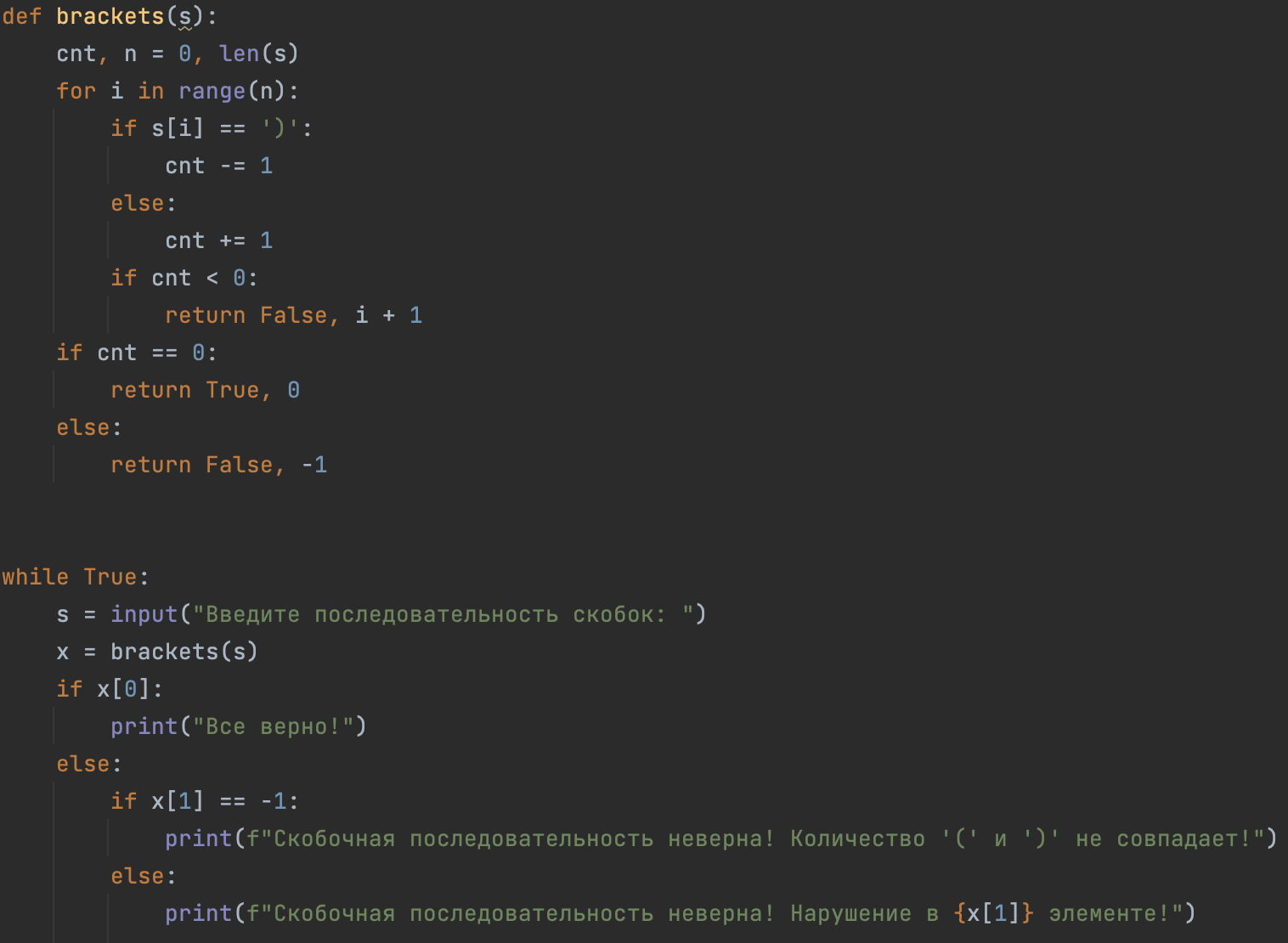


Рисунок 1 – Реализация функции, проверяющей правильность скобочной последовательности, и её тестирования

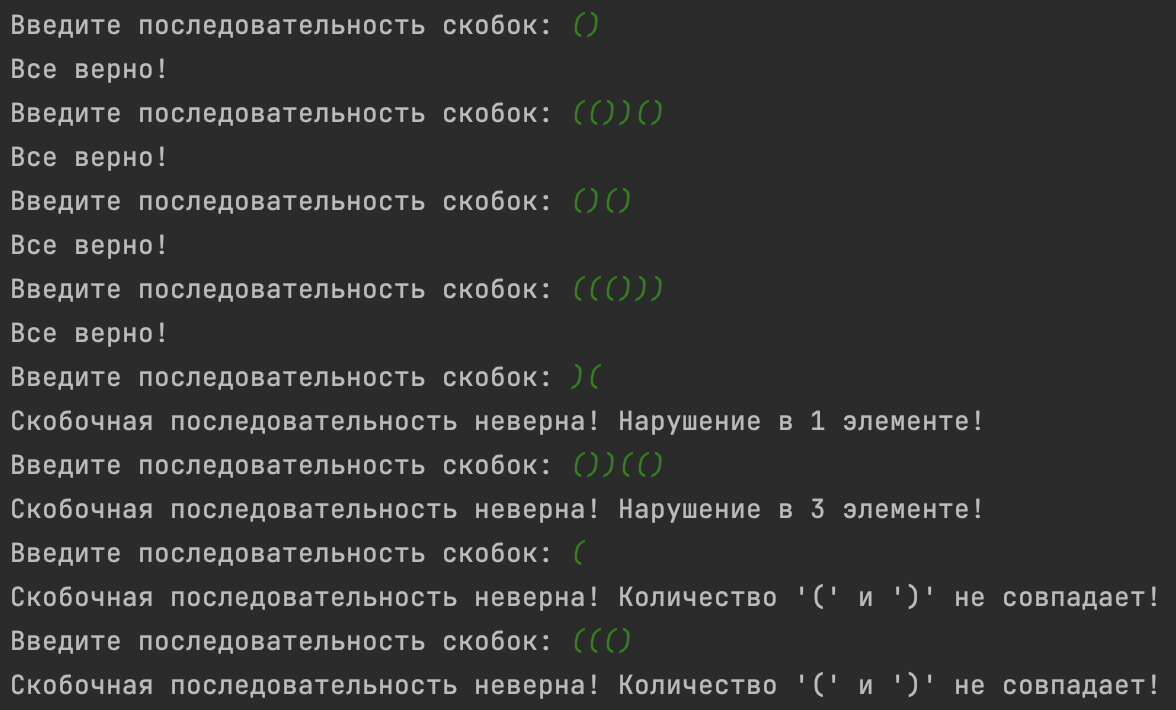


Рисунок 2 – Вывод программы, проверяющей правильность скобочной последовательности, для различных строк

# 

# 

# **Задание 2**

**Задание:** Придумайте и решите задачу для алгоритма поиска в глубину. Придумайте и решите задачу для алгоритма поиска в ширину. Объясните, почему для решения поставленных задач были выбраны именно эти алгоритмы поиска (подразумевается возможность выбора и других алгоритмов для решения поставленной задачи).

**Решение:**

1. Задача для алгоритма поиска в глубину

*Условие*: Дан ориентированный граф, представленный в виде списка смежности. Необходимо найти все вершины графа, достижимые из заданной стартовой вершины.

*Почему выбираем поиск в глубину?*

* Эффективность решения за счёт использования в алгоритме поиска рекурсии;
* Поиск в глубину обычно требует меньше памяти, так как он использует стек для хранения вершин, в то время как поиск в ширину использует очередь;
* Поиск в глубину обычно работает быстрее для графов с большим числом вершин и небольшой глубиной;
* Поиск в глубину просто реализовать.

*Реализация:*

Поиск в глубину реализует функция *dfs*, которая принимает граф (в виде словаря списков смежности), стартовую вершину и множество посещенных вершин (по умолчанию пустое множество). Функция начинает с посещения стартовой вершины, добавляет ее в множество посещенных и выводит ее на экран. Затем она проходит по всем соседним вершинам стартовой вершины и, если соседняя вершина еще не посещена, функция вызывается рекурсивно для этой вершины. В конце мы тестируем работу программы.

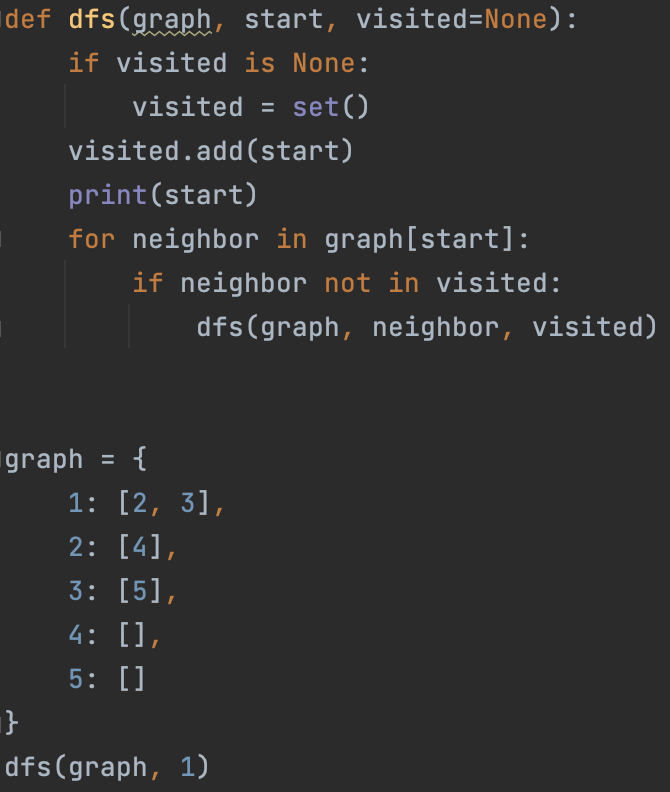


Рисунок 3 – Реализация программы, совершающей поиск достижимых из заданной вершины вершин используя поиск в глубину, и ее тестирование



Рисунок 4 – Вывод программы – вершины, в которые можно попасть из вершины 1

1. Задача для алгоритма поиска в ширину

*Условие*: Дан ориентированный невзвешенный граф без циклов. Необходимо найти кратчайший путь из заданной стартовой вершины до заданной конечной вершины

Пример решения на Python:

*Почему выбираем поиск в ширину?*

* Алгоритм поиска в ширину пошагово исследует все вершины на одном уровне от исходной вершины, прежде чем перейти на следующий уровень, что позволяет находить именно кратчайший путь;
* Алгоритм поиска в ширину гарантирует нахождение кратчайшего пути при поиске решения на невзвешенном графе без циклов.

*Реализация:*

Поиск в ширину реализуется функцией *bfs\_shortest\_path*. Данная функция принимает ориентированный граф в виде словаря, стартовую вершину и конечную вершину. Алгоритм инициализирует очередь *queue*, добавляет в нее путь, начинающийся с начального узла и продолжающийся по смежным узлам. Далее алгоритм начинает обходить граф в ширину, пока очередь не пуста. На каждой итерации алгоритм извлекает путь из начала очереди *queue.popleft()*, находит последний узел в этом пути *node* и проверяет, достигли ли мы конечного узла. Если да, то функция возвращает найденный путь. В противном случае алгоритм проходит по всем смежным узлам текущего узла *node*, создавая новые пути, добавляет их в очередь и продолжает поиск. Как только путь из начального узла до конечного узла будет найден, алгоритм завершит свою работу и вернет кратчайший путь в виде списка узлов. В конце мы тестируем работу программы.

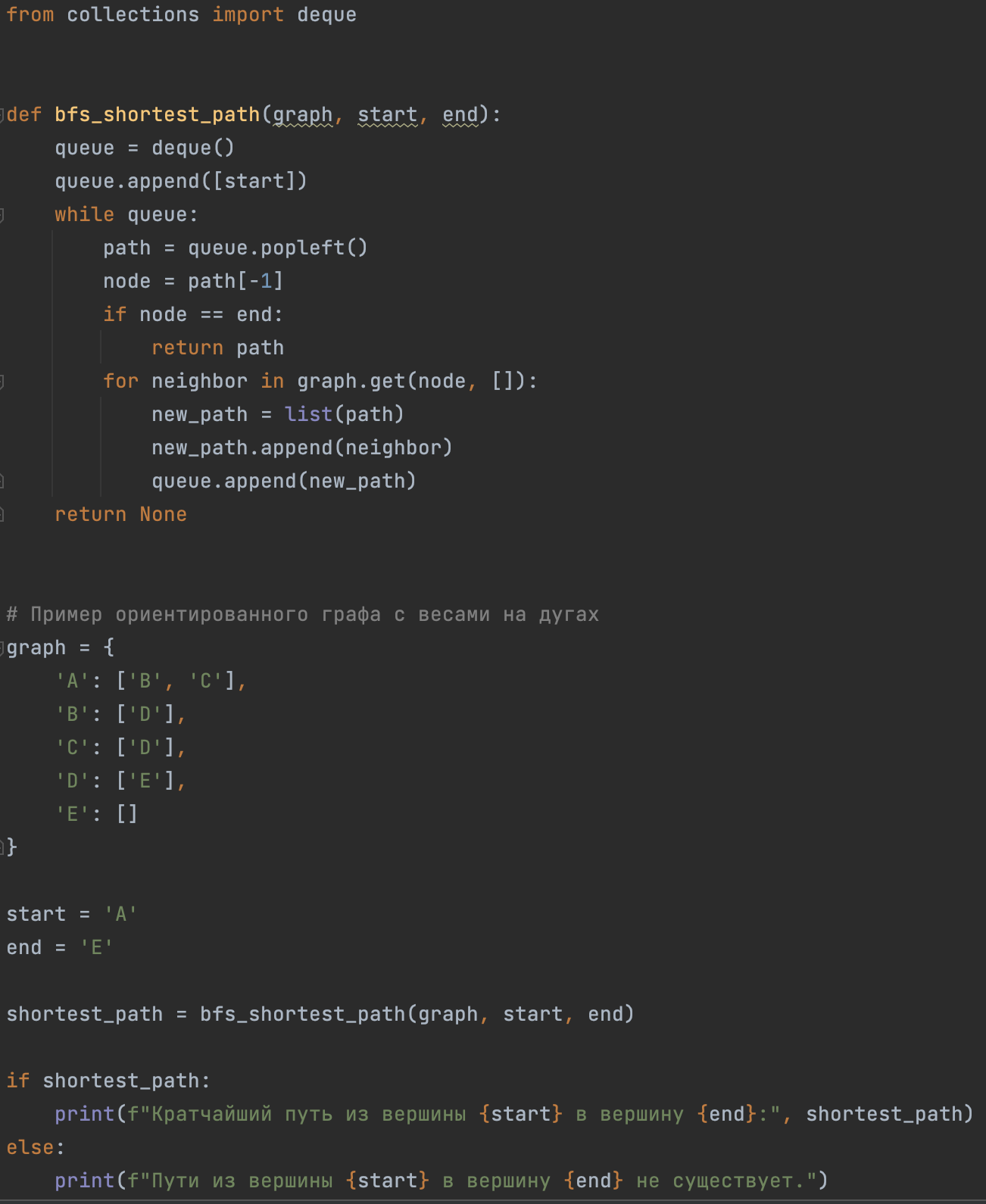


Рисунок 5 – Реализация функции, совершающей поиск кратчайшего пути между заданными вершинами используя поиск в ширину, и ее тестирование



Рисунок 6 – Вывод программы – кратчайший путь между вершинами A и E

# 

# **Задание 3**

**Задание:** Дана случайная квадратная матрица, заполненная нулями и единицами. Предположив, что 0 – это проход, а 1 – это стена, напишите алгоритм, который найдет выход из лабиринта.

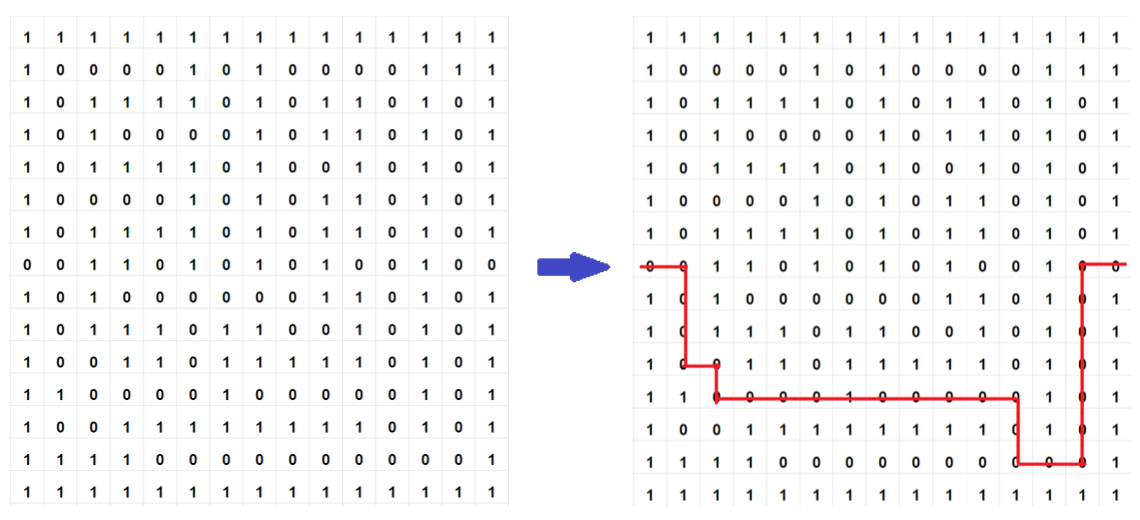


Рисунок 7 – Пример найденного выхода из лабиринта

**Решение:**

Для решения данной задачи воспользуемся прошлым заданием, а именно реализацией задачи нахождения кратчайшего пути из заданной стартовой вершины до заданной конечной вершины ориентированного невзвешенного графа без циклов. Задача поиска выхода из лабиринта очень на нее похожа – попробуем привести одну задачу к другой:

* В качестве имен вершин возьмем координаты нулей в массиве;
* Считаем, что друг с другом связаны все соседние вершины, а именно каждый конкретный ноль связан со своим верхним, нижним и правым соседом-нулем (при их наличии);
* В данном случае граф может обладать циклами (например, программа может бесконечно переходить от верхнего соседа к нижнему и наоборот), поэтому во избежание зацикливания программы добавим список, хранящий вершины, в которых мы уже были.

Реализуем программу. Алгоритм поиска кратчайшего маршрута для выхода из лабиринта реализуется с помощью функции *shortest\_path(mas, start)*. Для начала перейдем от двумерного массива к таблице смежности с помощью функции *make\_table(mas)*. Для каждой свободной клетки (то есть той клетки, в которой стоит 0) в таблице записываются все соседние свободные клетки в формате *(i, j)*, где *i* и *j* - координаты клетки.

Далее найдем потенциальные выходы из лабиринта (с учетом, что они находятся на противоположной входу стороне квадрата) – ими являются все нули из последнего столбца матрицы *mas*. Хранятся потенциальные выходы в массиве *ends*.

Сама функция *shortest\_path(mas, start)* использует алгоритм поиска в ширину для поиска кратчайшего пути между начальной точкой *start* и конечной точкой *end* в созданной таблице смежности. Во избежание зацикливания алгоритма вводим дополнительный массив *visited*, в котором хранятся те точки, которые мы уже посетили. Если путь найден, возвращается массив координат кратчайшего пути, если путь не найден, возвращается *None*.

Для примера задан двумерный массив mas (совпадает с примером на рисунке 7) и начальная точка.

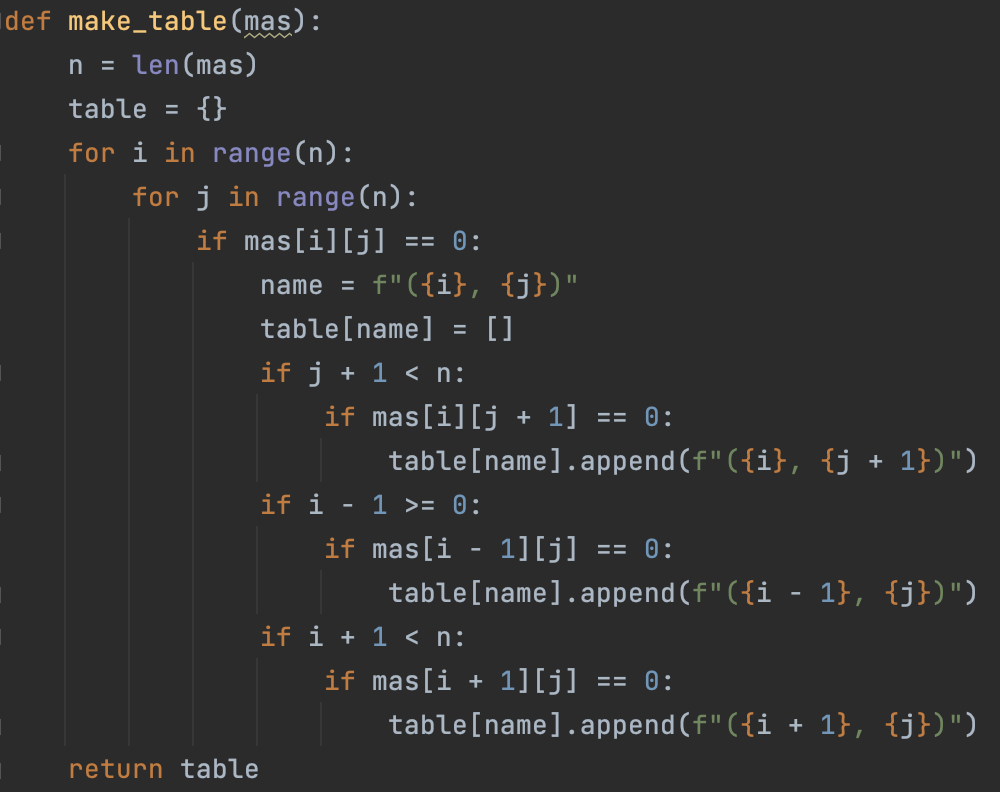


Рисунок 8 – Функция формирующая таблицу смежности из исходного массива

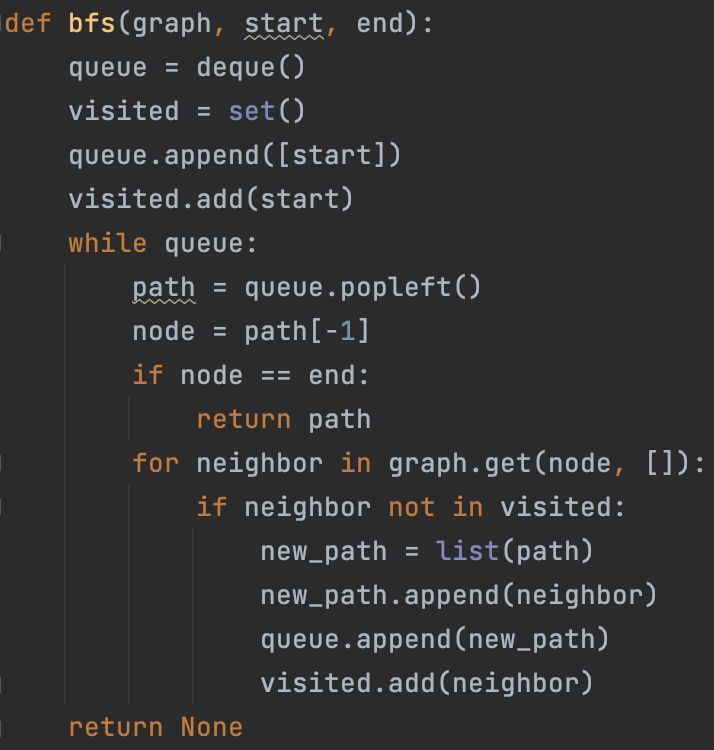


Рисунок 9 – Функция производящая обход в ширину

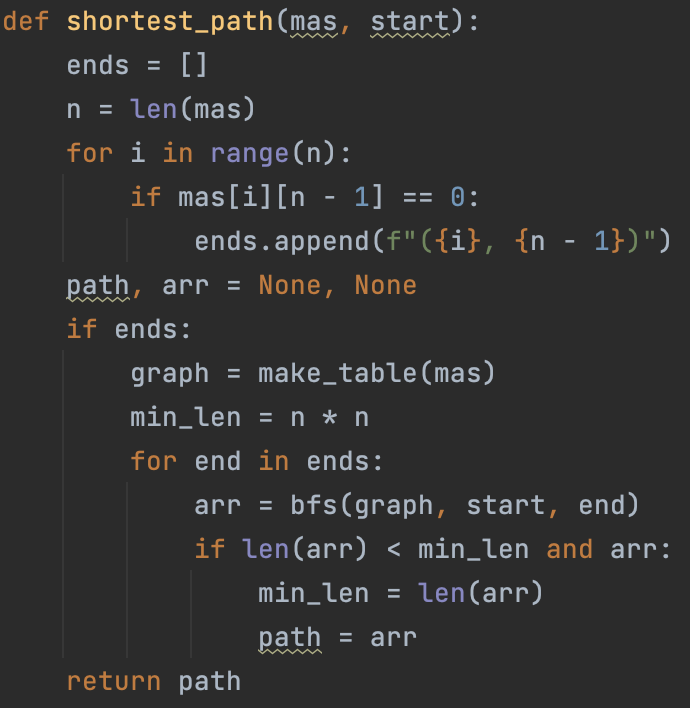


Рисунок 10 – Функция, ищущая кратчайший выход из лабиринта

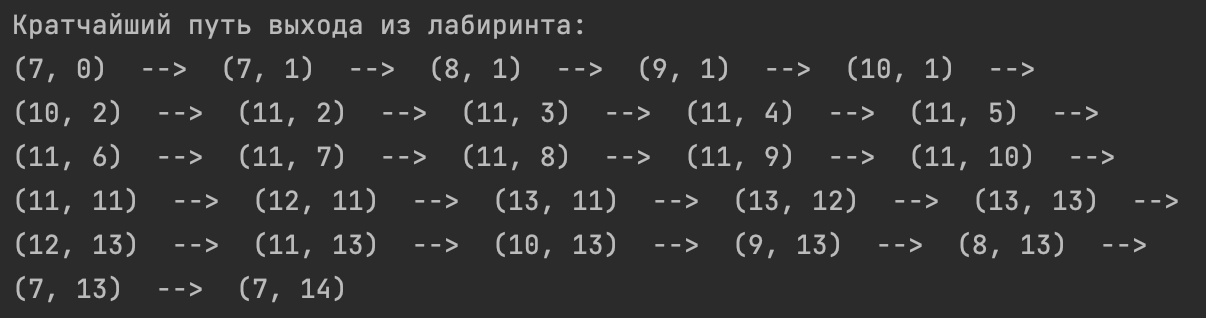


Рисунок 11 – Вывод программы для примера (рис. 1)

Из рисунка 11 видно, что написанная программа вывела тот же маршрут, что и предполагался.

# **ЗАКЛЮЧЕНИЕ**

В результате работы были разработаны программы для определения правильности скобочной структуры.

Также были придуманы и решены задачи с использованием алгоритмов поиска в глубину и поиска в ширину. Для каждого алгоритма поиска была найдена задача, для которой данный алгоритм является оптимальным. Были приведены аргументы в пользу использования алгоритма в придуманной задаче

В конце была написана программа для поиска выхода из лабиринта на матрице, заполненной нулями и единицами. Для этого был пересмотрен алгоритм поиска кратчайшего пути между двумя вершинами на невзвешенном графе, реализуемый с помощью поиска в ширину. Программа была успешно протестирована на примере.

Можно сказать, что изученные алгоритмы отлично подходят для эффективного решения задач, связанных с поиском и обходом графовых структур.

# **СПИСОК ЛИТЕРАТУРЫ**

1. Wikipedia. [Поиск в глубину](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D0%B2_%D0%B3%D0%BB%D1%83%D0%B1%D0%B8%D0%BD%D1%83). [Электронный ресурс] – (Дата последнего обращения 19.05.2024);
2. Wikipedia. [Поиск в ширину.](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D0%B2_%D1%88%D0%B8%D1%80%D0%B8%D0%BD%D1%83) [Электронный ресурс] – (Дата последнего обращения 19.05.2024).

# 

# **ПРИЛОЖЕНИЕ**

Для удобства все файлы выгружены на GitHub: <https://github.com/kathykkKk/Algorithms-and-Data-Structures-ICT.git>