УДК 004

Визуализация алгоритмов обхода графа с помощью Python и NetworkX

Романюк Виктория Дмитриевна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Фатеенков Данила Витальевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В рамках данной статьи рассмотрены алгоритмы обхода графов и с помощью модулей Matplotlib и NetworkX продемонстрирован процесс работы алгоритмов. Приведены код работы программ, а также описаны алгоритмы и их назначение. Также изучается связь математического аспекта работы с графами и рассматривается возможность использование созданных программ в процессе изучения теории графов.
Ключевые слова: Python, математика, теория графов, Matplotlib, NetworkX, обход в глубину, обход в ширину

Visualization of graph traversal algorithms using Python and NetworkX

Romanyuk Viktoriya Dmitrievna
Sholom-Aleichem Priamursky State University
Student

Fateenkov Danila Vitalievich
Sholom-Aleichem Priamursky State University
Student

Abstract
Within the framework of this article the algorithms of graph traversal are considered and with the help of Matplotlib and NetworkX modules the process of algorithms operation is demonstrated. The code of work of the programs is given, and also the algorithms and their purpose are described. The mathematical aspect of working with graphs is also studied and the possibility of using the created programs in the process of studying graph theory is considered.
Keywords: Python, math, graph theory, Matplotlib, NetworkX, depth traversal, width traversal

1. Введение

1.1 Актуальность

Алгоритмы обхода графов, такие как поиск в глубину (DFS) и поиск в ширину (BFS), являются основополагающими инструментами для анализа и обработки данных, представленных в виде графов. Они применяются для поиска путей, проверки связности, топологической сортировки, поиска остовных деревьев и решения задач на кратчайшие пути.

Также с развитием больших данных и сетевых технологий, необходимость эффективного анализа сложных сетевых структур становится все более острой. Например, в анализе социальных сетей графы представляют собой структуры взаимосвязей между пользователями. Алгоритмы обхода помогают выявлять кластеры пользователей, находить лидеров мнений и анализировать распространение информации.

Часто алгоритмы обхода графов используются в машинном обучении. Графовые нейронные сети (GNN) также основаны на принципах обхода и распространения информации по графу.

Все перечисленные причины актуальности данных алгоритмов свидетельствуют о том, что изучение методов обхода графов остаётся в настоящее время важным разделом в математике и правильный подход к визуализации данных алгоритмов может значительно упростить исследования и обучение других людей.

1.2 Обзор исследований

В.В. Воронин и Г.И. Бахрушина в своей работе рассмотрели поиск кратчайшего пути во взвешенном графе с помощью алгоритма Беллмана-Форда, который лежит в основе протоколов маршрутизации RIP и BGP [1]. Была также разработана программа на языке python 3.52, реализующая данный алгоритм.

Г. Д. Секачев описал процесс выбора наиболее эффективного алгоритма визуализации результата обработки направленного цикличного графа, на примере задачи поиска наибольшего простого пути [2].

В. А. Моисеенко и Е. В. Комракова провели исследование теории графов, а также решили задачу, которая связана с исследованием алгоритмов и применением их в нужных ситуациях [3].

А. И. Якимов, И. В. Харламов и В. А. Шунькин разработали программы для реализации алгоритма поиска в ширину и алгоритма поиска в глубину на языке программирования Python для изучения теории графов [4].

М. А. Давыдовский описал модификацию алгоритма Дейкстры поиска кратчайших путей в графе, заключающаяся в выборе на очередном шаге алгоритма только одного маршрута, проходящего через вершину [5].

1.3 Цель исследования

Цель – создать программу для визуализации алгоритмов обхода графа в глубину и ширину с пошаговым представлением работы алгоритмов.

2 Материалы и методы

Для реализации поставленных задач используется язык программирования Python, а также модули Matplotlib и NetworkX.

3 Результаты и обсуждения

Обход в глубину (DFS, Depth-First Search) – это один из классических алгоритмов для обхода графов, который исследует вершины и ребра графа, погружаясь по каждому пути как можно глубже, прежде чем вернуться назад и начать исследование других путей. Он применяется для поиска всех вершин графа, достижимых из начальной вершины, и решает ряд важных задач, таких как поиск компонент связности, топологическая сортировка, проверка наличия циклов и др.

Алгоритм DFS работает на основе рекурсии или с использованием стека, что обеспечивает следование по ребрам графа до тех пор, пока возможно двигаться глубже. Если дальнейшее продвижение по текущему пути невозможно (то есть все смежные вершины уже посещены), алгоритм возвращается назад и начинает исследование других путей.

В свою же очередь поиск в ширину (Breadth-First Search, BFS) используется для изучения всех узлов и ребер графа. Он работает по принципу "уровневого" обхода, исследуя сначала все узлы на текущем уровне, прежде чем переходить к узлам на следующем уровне. Этот алгоритм особенно полезен для решения задач, связанных с нахождением кратчайших путей и определением связности графа.

Для реализации BFS используется очередь, которая обеспечивает доступ к элементам в порядке их добавления. Узлы графа добавляются в очередь по мере их обнаружения и извлекаются из очереди по мере обработки.

Для визуализации описанных алгоритмов можно использовать языка программирования Python и 2 дополнительных модуля:

1. Matplotlib – модуль для построения графиков и визуального отображения математических данных.
2. NetworkX – модуль для работы с графами. Содержит в себе необходимые функции для обхода графов.

В программе были инициализированы оба модуля следующим образом:

```
import matplotlib.pyplot as plt
import networkx as nx
```

Также для демонстрации алгоритмов и их разницы можно представить работу визуализации пошагово. Для этого необходимо подключить подмодуль animation, который также находится в Matplotlib. В контексте данной работы интересен только класс FuncAnimation, который позволяет обновлять граф на каждом шаге. Он принимает функцию update, которая выполняется для каждого кадра анимации:

```
from matplotlib.animation import FuncAnimation
```

В первую очередь необходимо инициализировать сам граф в коде программы и добавить рёбра. Граф в программе сохранён в переменную G и представляет собой экземпляр класса nx.Graph(), а его рёбра представлены в виде массива (переменная для хранения массива называется edges) кортежей из двух элементов, которые представляют собой номера узлов, которые нужно соединить. Для добавления узлов и рёбер в граф необходимо воспользоваться функцией add_edges_from, которая работает с экземплярами класса nx.Graph() и в качестве аргумента принимает массив edges:

```
G = nx.Graph()
edges = [(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6), (6,4), (3,4)]
G.add_edges_from(edges)
```

На данном этапе можно визуализировать граф, но никакого обхода нет, поэтому все рёбра будут окрашены в чёрный цвет (см. рис. 1). Для вывода графа на экран необходимо вычислить позиции узлов на плоскости для дальнейшей визуализации. Этот процесс может быть автоматизирован с помощью модуля NetworkX. В нём представлен ряд функций позиционирования узлов:

1. nx.spring_layout: использует физическую модель для расположения узлов.
2. nx.circular_layout: располагает узлы по окружности.
3. nx.random_layout: использует алгоритмы псевдослучайных чисел для расположения узлов на экране.
4. nx.shell_layout: узлы располагаются по концентрическим окружностям.
5. nx.spectral_layout: использует линейную алгебру для определения позиций, основанных на спектральных свойствах графа.
6. nx.kamada_kawai_layout: второй вариант силовой модели для создания равномерно распределённого графа.

Для визуализации был выбран 6-й вариант распределения узлов на экране. Помимо определения расположения узлов необходимо инициализировать объект plt.subplots, который создаёт объект Figure (фигура) и один или несколько объектов Axes (ось, на которой можно рисовать графики). После этапа инициализации достаточно только отобразить граф на экране с помощью функции draw(). Важно отметить – при каждом повторном запуске отображение графа будет меняться, поэтому на последующих рисунках расположение узлов может изменяться, но рёбра останутся неизменными:

```
pos = nx.kamada_kawai_layout(G)
fig, ax = plt.subplots()
```

```
    nx.draw(G, pos, with_labels=True, node_color='lightblue',
node_size=500, font_size=10, ax=ax)
    plt.show()
```

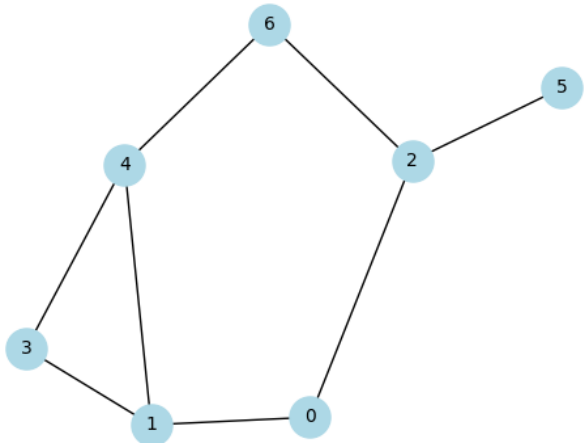

Рисунок 1. Визуализация графа с помощью Python и Matplotlib

На данном этапе нельзя определить работу алгоритмов обхода в ширину и глубину, так как он ещё не инициализирован. Для алгоритма обхода в глубину необходимо воспользоваться функцией nx.dfs_edges(), которая анализирует построенный граф и возвращает список рёбер, по которым был произведён обход. В качестве аргумента данная функция принимает одно значение – экземпляр класса nx.Graph(), у которого заранее были объявлены узлы и рёбра. Этот метод часто применяется для различных задач обхода графов, таких как поиск всех путей, тестирование связности, иерархический анализ, а также для построения дерева поиска. Также вторым аргументом данная функция принимает номер узла, с которого необходимо начинать обход графа:

```
    dfs_edges = list(nx.dfs_edges(G, source=0))
```

Например, для определённого в программе ранее графа результат работы данной функции будет следующим: [(0, 1), (1, 3), (3, 4), (4, 6), (6, 2), (2, 5)].

На основе полученного списка можно пошагово визуализировать обход графа. Для этого был определён пустой список current_edges, в который будут записываться грани, которые уже были пройдены алгоритмом. Также была определена функция update(), которая каждую секунду будет обновлять изображение на экране, тем самым визуализируя работу алгоритма обхода:

```
    def update(num):
        ax.clear()
        nx.draw(G,              pos,              with_labels=True,
node_color='lightblue', node_size=500, font_size=10, ax=ax)
        current_edges.append(dfs_edges[num])
        nx.draw_networkx_edges(G, pos, edgelist=current_edges,
edge_color='r', width=2, ax=ax)
```

```
        ax.set_title(f'Шаг {num+1} из {len(dfs_edges)}')

    ani = FuncAnimation(fig, update, frames=len(dfs_edges),
interval=1000, repeat=False)
```

Данный алгоритм работает следующим образом: ax.clear() очищает подграф ax для того, чтобы на каждом шаге анимации рисовать граф заново, обновляя его состояние. Затем рисуется весь граф G с заданными параметрами (положение узлов pos, цвет узлов lightblue, размер узлов 500, размер шрифта меток 10). Важно отметить, что граф рисуется заново, но пока без выделения рёбер, которые уже были пройдены в процессе обхода. На текущем этапе также происходит добавление пройденного шага из списка dfs_edges в список current_edges, а затем на графе отображается ребро красным цветом, которое прошёл алгоритм на данном шаге. Также для удобства был добавлен текст поверх графа, чтобы можно было идентифицировать текущий шаг.

Последний шаг – запуск алгоритма обновления изображения на экране с помощью функции FuncAnimation.

Результат работы программы: пошаговое отображение обхода графа в глубину на экране (см. рис. 2).

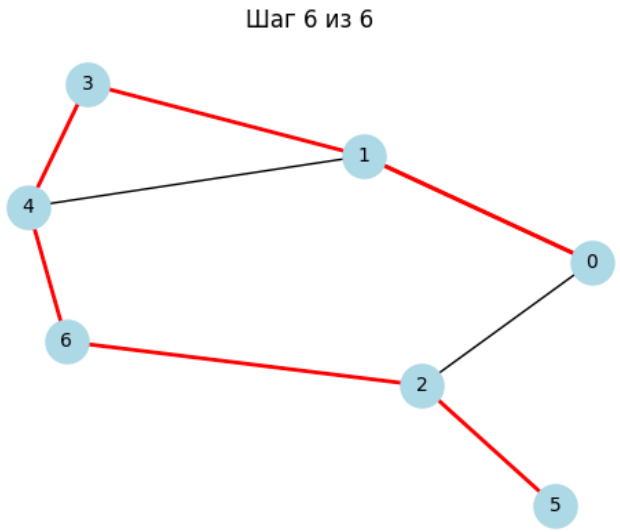

Рисунок 2. Визуализация результата обхода графа в глубину с помощью Python и Matplotlib

Для большей наглядности можно также вывести каждый шаг работы алгоритма на экран. В таком случае код программы незначительно изменится: будет использоваться режим отображения plt.tight_layout() и вместо пошагового перерисовывания графов на экран будут выводиться все шаги друг за другом (см. рис. 3).

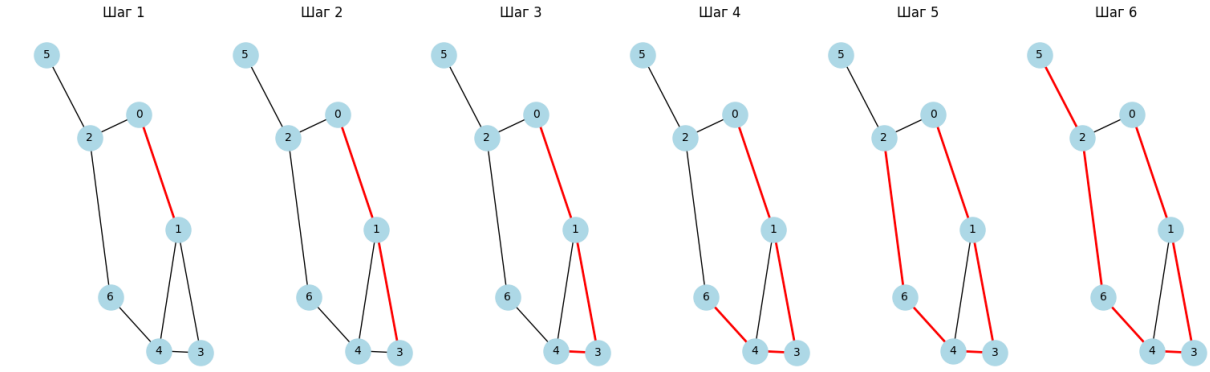

Рисунок 3. Пошаговое отображение алгоритма обхода графа в глубину

Визуализировать алгоритм обхода в ширину можно по такому же принципу, но изменится функция для расчёта маршрута обхода графа:

```
bfs_edges = list(nx.bfs_edges(G, source=0))
```

В случае данного алгоритма результат работы функции будет следующим: [(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6)]. Не трудно заметить разницу в работе описанных алгоритмов. Разницу также можно увидеть и при пошаговом представлении работы алгоритма в ширину (см. рис. 4).

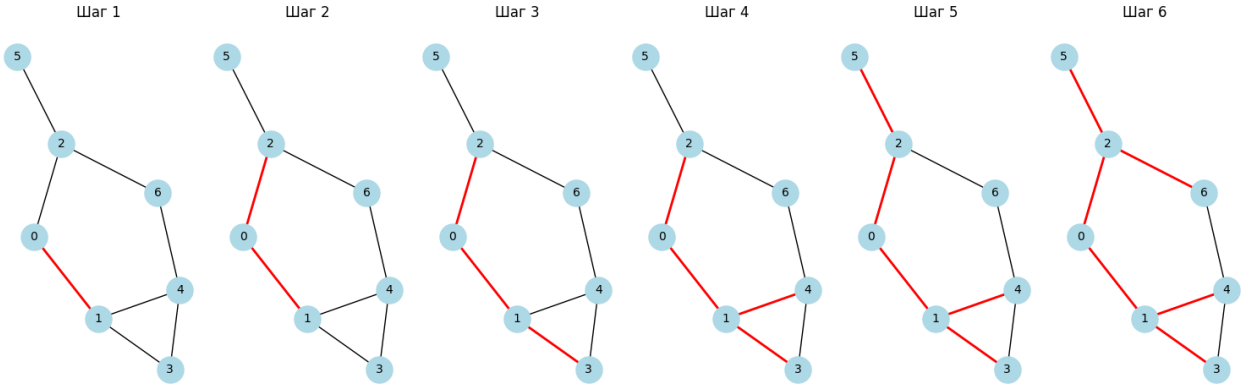

Рисунок 4. Пошаговое отображение алгоритма обхода графа в ширину

В библиотеке NetworkX можно использовать различные методы обхода графов, помимо обхода в глубину и ширину:

1. Поиск кратчайшего пути: для этого нужно использовать специальную функцию nx.shortest_path(), которая позволяет найти кратчайший путь от заданной вершины до всех других вершин.
2. Обход в глубину с ограничением по длине пути: для данного алгоритма есть функция nx.edge_dfs(), которая производит обход графа с учётом направления рёбер и возможным ограничением глубины поиска.
3. Алгоритм Дейкстры: находит кратчайший путь между двумя вершинами, нужна функция nx.dijkstra_path().
4. Алгоритм Беллмана-Форда: данный алгоритм работает с отрицательными весами рёбер, реализован в виде функции nx.bellman_ford_path().

5. Алгоритм A*: это алгоритм поиска пути, который использует эвристику для ускорения поиска, нужна функция nx.astar_path().

6. Поиск коннектед-компонент: для выполнения данной задачи нужна функция nx.connected_components(), которая находит все компоненты связности в неориентированном графе.

Для наглядности также можно вывести не только каждый шаг работы алгоритма, но и отдельно ребро, которое задействуется в обходе на данном шаге (см. рис. 5). Такой подход к визуализации позволяет более наглядно продемонстрировать принцип работы алгоритмов обхода графов.

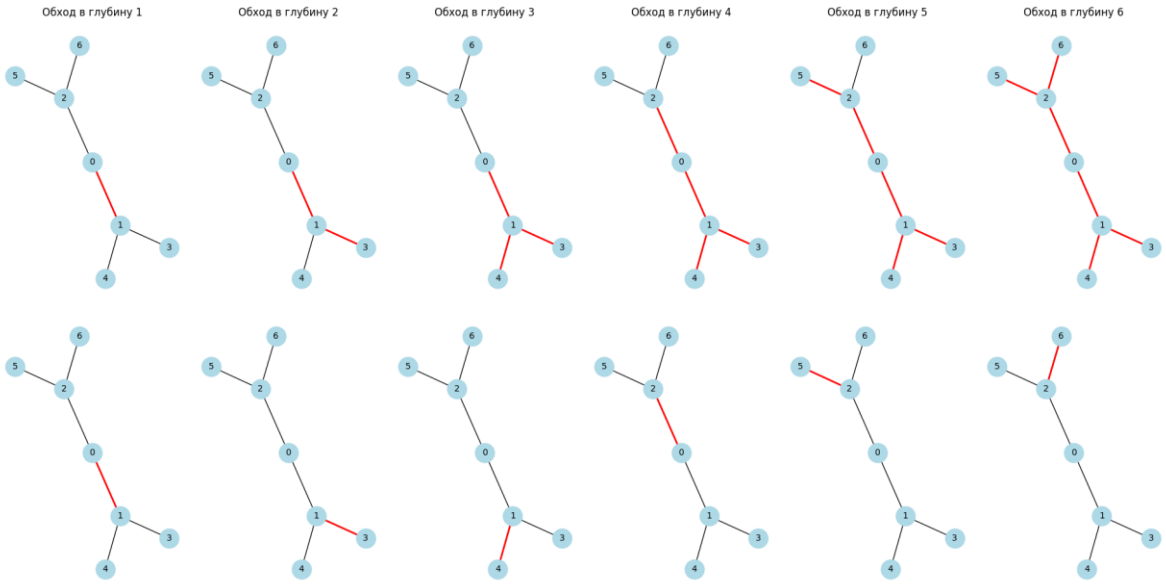

Рисунок 5. Детальное представление работы алгоритма обхода графа в глубину

Python предоставляет весь необходимый функционал для работы с графами в виде отдельных модулей и функций, что делает его хорошим инструментом для организации обучения учащихся разных уровней образования. Удобство применения Python обусловлено не только наличием необходимых функций, но и также простотой синтаксиса и применения, что делает его отличным выбором, если необходимо продемонстрировать работу алгоритмов обхода графов наглядным образом.

4 Выводы

В рамках данной статьи был представлен код программы визуализации алгоритмов обхода неориентированного и невзвешенного графа на языке программирования Python с использованием Matplotlib и NetworkX. Код программы может быть расширен за счёт добавления возможности работы с ориентированными и взвешенными графами.

Библиографический список

1. Воронин В.В., Бахрушина Г.И. Поиск кратчайшего пути во взвешенном орграфе с помощью алгоритма Беллмана-Форда // В сборнике: Far East Math. Материалы студенческой национальной научной конференции. Хабаровск, 2021. С. 53-55.
2. Секачев Г. Д. Поиск наилучшего алгоритма визуализации графа (на примере результата работы алгоритма поиска наибольшего пути) // Общество. 2024. № 2-1(33). С. 108-114.
3. Моисеенко В. А., Комракова Е. В. Решение графовых задач при помощи Python // Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях: Материалы XXIV Республиканской научной конференции студентов и аспирантов, Гомель, 22–24 марта 2021 года / Министерство образования Республики Беларусь; Учреждение образования «Гомельский государственный университет имени Франциска Скорины». Гомель: Гомельский государственный университет им. Франциска Скорины, 2021. С. 356-357.
4. Якимов А. И., Харламов И. В., Шунькин В. А. Изучение теории графов с применением языка программирования Python // Современные вопросы естествознания и экономики: сборник трудов V Международной научно-практической конференции, Прокопьевск, 16 марта 2023 года. Прокопьевск: Филиал Федерального государственного бюджетного образовательного учреждения высшего профессионального образования "Кузбасский государственный технический университет имени Т. Ф. Горбачева" в г. Прокопьевске, 2023. С. 13-16.
5. Давыдовский М. А. Модификация алгоритма поиска кратчайших путей Дейкстры на основе выбора одного маршрута // Цифровая трансформация транспорта: проблемы и перспективы: материалы III Международной научно-практической конференции, Москва, 27 сентября 2023 года. Москва: Российский университет транспорта, 2023. С. 303-307.