ISSN 2225-9309 (Print)
2025. № 3 (34). С. 126–130

Молодежный
Вестник УГАТУ

http://journal.ugatu.su

УДК 519.178

doi 10.54708/22259309_2025_334126

ПО ДЛЯ ЗАДАЧИ НАХОЖДЕНИЯ МИНИМАЛЬНОГО ПУТИ ВО ВЗВЕШЕННОМ ГРАФЕ

Д. А. Ашихмин[1]

[1]ashikhmin.dima@inbox.ru

ФГБОУ ВО «Уфимский университет науки и технологий» (УУНиТ)

Аннотация. В статье рассматривается алгоритм Дейкстры, используемый для нахождения минимального пути во взвешенных графах. Исследованы теоретические основы и практические методы применения алгоритма. Представлена программная реализация алгоритма.

Ключевые слова: алгоритм Дейкстры; граф; взвешенный граф; реализация алгоритма; Python; минимальный путь; кратчайший путь.

ВВЕДЕНИЕ

Алгоритм, получивший имя своего автора – нидерландского исследователя Эдсгера Дейкстры, был предложен в 1959 г. с целью эффективного решения задачи поиска кратчайшего пути в графах, что имело важное значение для оптимизации маршрутов и сетевых коммуникаций. Его основная задача заключается в определении минимального расстояния от выбранной начальной вершины до всех остальных вершин графа. Изначально алгоритм предназначался для работы со взвешенными графами, причём особенно хорошо он проявляет себя в случаях, когда веса рёбер являются неотрицательными. Хотя изначально применялся к неориентированным графам, существуют и версии, адаптированные для ориентированных структур.

ШАГИ АЛГОРИТМА

В рамках алгоритма Дейкстры каждой вершине из множества V присваивается числовой показатель – он отражает кратчайшее известное на данный момент расстояние от начальной вершины v до данной. По сути, это предполагаемая «стоимость» пути от старта до конкретной точки графа.

На старте выполнения метка вершины v устанавливается равной нулю, поскольку путь к самой себе не требует затрат. Все остальные вершины получают метки, равные бесконечности – они пока недоступны. Параллельно с этим все вершины помечаются как ещё не исследованные.

Алгоритм работает итеративно. На каждом шаге выбирается одна из непосещённых вершин с наименьшей текущей меткой – пусть это будет вершина u. Далее проверяются все её соседние вершины, то есть такие, с которыми u напрямую соединена ребром. Для каждого такого соседа рассчитывается потенциальная длина пути от стартовой вершины через u. Если новая длина оказывается меньше той, что записана у соседа, то метка обновляется – это означает, что найден более короткий маршрут.

После обработки всех соседей вершина u считается посещённой, и в дальнейших шагах в расчётах участвовать уже не будет. Алгоритм повторяет этот процесс, пока не будут обработаны все вершины графа.

Стоит отметить, что первой вершиной в алгоритме всегда будет изначальная вершина, в нашем случае вершина v. В конце алгоритма на каждой метке, за исключением метки вершины v (так как вершина v является начальной, то и путь до нее будет самым минимальным и останется тем же самым – 0), будет обновленное значение (за исключением тех вершин, до которых нет ни прямого, ни косвенного пути, их метки будут равны бесконечности, что означает, что добраться до них из вершины v невозможно). Новые значения меток будут равны минимальному пути из вершины v до выбранных вершин.

Алгоритм Дейкстры широко используется для поиска кратчайших путей в графах с неотрицательными весами. Его идея проста и эффективна: определить наименьшую «стоимость» перемещения от одной вершины ко всем остальным, учитывая веса рёбер. Благодаря универсальности и надёжности алгоритм нашёл применение в самых разных сферах.

Одним из наиболее распространённых примеров его использования являются навигационные системы. Такие приложения, как Google Maps или Яндекс Навигатор, строят маршрут не просто по расстоянию, но и принимая во внимание дорожную обстановку, пробки, ремонтные работы и другие условия. В результате, пользователь получает не только кратчайший путь, но и наиболее удобный по времени, что особенно важно в условиях городского трафика.

В сфере сетевых технологий алгоритм также играет важную роль. Например, в протоколе OSPF он помогает находить наилучшие маршруты для передачи данных между узлами. Это обеспечивает более стабильную и быструю работу компьютерных сетей, даже если их структура достаточно сложная.

Также алгоритм активно применяется в логистике. Его используют для расчёта оптимальных маршрутов доставки, особенно когда необходимо минимизировать затраты на транспорт или ускорить перемещение между точками. Такой подход эффективен как в масштабах городских перевозок, так и внутри крупных складских комплексов.

Интересно, что алгоритм активно используется и в навигации автономных устройств – роботов, дронов и даже автоматизированных тележек. Им важно не просто доехать до цели, а сделать это безопасно и с наименьшими затратами энергии. Здесь Дейкстра действительно показывает свою практическую силу.

Что касается компьютерных игр, особенно стратегий и RPG, то автор лично часто замечал, как персонажи двигаются не просто «по прямой», а выбирают разумный маршрут. Это тоже результат работы алгоритмов, подобных Дейкстре, и это делает поведение игровых объектов более реалистичным.

Также нельзя не упомянуть применение в финансовых и социальных моделях. Алгоритм помогает находить самые короткие связи между компаниями или пользователями, и на основе этого можно строить рекомендации или оценивать, кто на кого больше влияет.

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Чтобы лучше разобраться, как работает алгоритм, мы написали его программную реализацию. В качестве языка выбрали Python – он простой, удобный, и у него много полезных библиотек, особенно для работы с графами. В частности, использовали библиотеку NetworkX для создания и обработки графов, а также Matplotlib – с её помощью удобно визуализировать результат. На рис. 1 показано подключение этих библиотек.

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

```

Рис. 1 Networkx и Matplotlib

На рис. 2 представлена сама функция для нахождения минимального пути – алгоритм Дейкстры.

```
def AlgDijk(a, b, G):  1 usage
    G.nodes[a]['metka'] = 0
    G.nodes[a]['isVis'] = False
    prev = {}
    for node in G.nodes:
        if node != a:
            G.nodes[node]['metka'] = float('inf')
            G.nodes[node]['isVis'] = False
    while True:
        if all(G.nodes[n]['isVis'] for n in G.nodes):
            break
        min_metka = float('inf')
        chosen_node = None
        for nod in G.nodes:
            if not G.nodes[nod]['isVis'] and G.nodes[nod]['metka'] < min_metka:
                min_metka = G.nodes[nod]['metka']
                chosen_node = nod
        if chosen_node is None:
            break
        G.nodes[chosen_node]['isVis'] = True
        for neighbor in G.neighbors(chosen_node):
            if not G.nodes[neighbor]['isVis']:
                weight = G[chosen_node][neighbor]['weight']
                new_metka = G.nodes[chosen_node]['metka'] + weight
                if new_metka < G.nodes[neighbor]['metka']:
                    G.nodes[neighbor]['metka'] = new_metka
                    prev[neighbor] = chosen_node  # сохраняем путь
    if G.nodes[b]['metka'] == float('inf'):
        return float('inf'), "Путь не существует"
    path = []
    current = b
    while current != a:
        path.append(current)
        current = prev[current]
    path.append(a)
    path.reverse()
    route_str = "->".join(map(str, path))
    return G.nodes[b]['metka'], route_str
```

Рис. 2 Функция AlgDijk

На рис. 3 продемонстрировано создание небольшого консольного интерфейса.

```
G = nx.Graph()
key = 0
while True:
    print("Введите команду\n"
          "1. Создать вершину\n"
          "2. Создать ребро\n"
          "3. Закончить с созданием")
    key = int(input())
    if key == 1:
        count_of_node = len(G.nodes) +1
        G.add_node(count_of_node)
    elif key == 2:
        print("Введите первую вершину:")
        frst_nd = int(input())
        print("Введите вторую вершину:")
        scnd_nd = int(input())
        print("Введите расстояние между вершинами:")
        wght = int(input())
        G.add_edge(frst_nd, scnd_nd, weight=wght)
    elif key == 3:
        break

print("Введите искомый минимальный путь из вершины A в вершину B")
a = int(input("A: "))
b = int(input("B: "))
if not a in G :
    print(f"В графе нет вершины {a}")
if not b in G :
    print(f"В графе нет вершины {b}")
min_way,route = AlgDijk(a,b,G)
if min_way == float('inf'):
    print("Пути между вершинами не существует!")
else:
    print(f"минимальный путь из вершины A в вершину B составляет {min_way}, с таким маршрутом {route}")
```

Рис. 3 Реализация консольного интерфейса

Далее на рис. 4 идет работа уже с визуализацией графа.

```

pos = nx.spring_layout(G, seed=42)
edge_labels = nx.get_edge_attributes(G, name: 'weight')
plt.figure(figsize=(8, 6))
nx.draw(
    G, pos,
    with_labels=True,           # показывать номера вершин
    node_color='lightblue',     # цвет вершин
    node_size=1000,             # размер вершин
    font_size=12,               # размер шрифта внутри вершин
    edge_color='gray',          # цвет рёбер
    width=2                     # толщина рёбер
)
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.title("Визуализация графа с весами рёбер")
plt.axis('off')
plt.tight_layout()
plt.show()
```

Рис. 4 Визуализация графа

Касательно скорости выполнения алгоритма Дейкстры можно сказать следующее. В представленной реализации используется классический подход, при котором на каждом шаге производится перебор всех непосещённых вершин с целью нахождения той, у которой минимальное значение метки. Такой способ не использует продвинутых структур данных, таких как очередь с приоритетом или куча, и, по сути, повторяет наивную версию алгоритма. В результате, сложность алгоритма составляет $O(V^2)$, где V – количество вершин в графе. Это может быть вполне допустимо при работе с небольшими графами. Однако при увеличении числа вершин такой подход становится всё менее оправданным, поскольку на обработку одного лишь выбора вершины с минимальной меткой затрачивается значительное количество операций.

Для улучшения производительности алгоритма при работе с крупными графами применяется приоритетная очередь, реализованная, например, через бинарную или фибоначчиеву кучу. В этом случае сложность снижается до $O((V + E) \log V)$, где E – количество рёбер. Такая реализация позволяет значительно ускорить выполнение, особенно на разреженных графах, где число рёбер существенно меньше квадрата количества вершин. Помимо ускорения самой логики выбора вершины с минимальной меткой, также облегчается пересчёт расстояний до соседей, так как приоритетная очередь обеспечивает быстрый доступ к вершинам с минимальными значениями.

Таким образом, при решении прикладных задач, особенно в таких сферах, как навигационные системы, маршрутизация в сетях и логистика, необходимо учитывать масштаб графа и выбирать наиболее подходящую реализацию алгоритма. В условиях ограниченных ресурсов предпочтение следует отдавать более эффективным структурам данных. Реализация алгоритма Дейкстры на языке Python с использованием библиотеки NetworkX демонстрирует его основные принципы, что особенно полезно при изучении.

ЗАКЛЮЧЕНИЕ

Алгоритм Дейкстры считается жадным, потому что на каждом шаге он выбирает вершину с наименьшей текущей меткой, то есть минимальным известным расстоянием от начальной вершины, и сразу помечает её как посещённую, не проверяя, возможно ли в будущем найти более короткий путь. Он принимает локально оптимальное решение, рассчитывая, что оно приведёт к глобально оптимальному результату. Такой подход работает только в графах с неотрицательными весами, потому что отрицательные значения могут привести к тому, что более выгодный путь откроется позже, но алгоритм уже прошёл вершину и не вернётся. Именно за это поведение – стремление ко мгновенной выгоде на каждом шаге – алгоритм и называют жадным.

СПИСОК ЛИТЕРАТУРЫ

1. Черемисин В. А. Алгоритмы на графах: Учебное пособие / В. А. Черемисин. М.: МГТУ им. Н. Э. Баумана, 2021. 128 с.
2. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн; пер. с англ. 3-е изд. М.: Вильямс, 2022. 1328 с.
3. Поречный С. С., Житникова Н. И., Шерыхалина Н. М., Ураков А. Р. Дискретная математика / С. С. Поречный, Н. И. Житникова, Н. М. Шерыхалина, А. Р. Ураков. Уфа: РИК УГАТУ, 2019. 400 с.

ОБ АВТОРАХ

АШИХМИН Дмитрий Алексеевич, студент ПРО ИИМРТ УУНиТ.

METADATA

Title: Software for the problem of finding the minimum path in a weighted graph.

Author: D. A. Ashikhmin[1]

Affiliation:

[1] Ufa University of Science and Technology (UUST), Russia.

Email: [1]ashikhmin.dima@inbox.ru

Language: Russian.

Source: Molodezhnyj Vestnik UGATU (scientific journal of Ufa University of Science and Technology), no. 3 (34), pp. 126-130, 2025. ISSN 2225-9309 (Print).

Abstract: The article discusses Dijkstra's algorithm used to find the minimum path in weighted graphs. The theoretical foundations and practical methods of applying the algorithm are investigated. A software implementation of the algorithm is presented.

Key words: Dijkstra's algorithm, graph, weighted graph, algorithm implementation, Python, minimum path, shortest path.

About authors:

ASHIKHMIN, Dmitriy Alekseevich, student PRO IIMRT UUST.