МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ

**«Национальный исследовательский технологический университет «МИСИС»**

**Институт Компьютерных Наук**

**Отчет**

**Алгоритм построения кратчайших путей на сети с единичными длинами.**

**По курсу:** Комбинаторика и теория графов

**Ссылка на репозиторий:**

<https://github.com/aegon-7n/combinatorics-and-graph.git>

Трушков Глеб Викторович

Группа БИВТ-23-6

**Содержание**

1. Формальная постановка задачи
2. Теоретическое описание алгоритма и его характеристики
3. Сравнительный анализ с аналогичными алгоритмами
4. Перечень инструментов, используемых для реализации
5. Описание реализации и процесса тестирования
6. Заключение

**1. Формальная постановка задачи**

**Задача**:  
Построение кратчайших путей в ориентированном графе, где длина каждого рёбра равна единице. Требуется найти расстояние от заданной вершины (истока) до всех остальных вершин графа.

**Входные данные**:

* Ориентированный граф G = (V, E), где:
  + V — множество вершин;
  + E — множество рёбер с длиной рёбер, равной 1 для каждого ребра.
* Начальная вершина s ∈ V.

**Выходные данные**:

* Массив расстояний dist, где dist[v] — это кратчайшее расстояние от вершины s до вершины v.

**2. Теоретическое описание алгоритма и его характеристики**

**Описание алгоритма BFS**:

Алгоритм поиска в ширину (BFS) эффективно решает задачу нахождения кратчайших путей в графах с единичными рёбрами. Основные шаги алгоритма:

1. **Инициализация**: Все вершины, кроме начальной, получают расстояние ∞, начальная вершина s имеет расстояние 0.
2. **Обход в ширину**:
   * Начинаем с вершины s, помещаем её в очередь.
   * Для каждой вершины u, извлеченной из очереди, перебираем её соседей v, и если для соседа v не установлено расстояние, то его расстояние будет dist[v] = dist[u] + 1, и добавляем его в очередь.
3. **Завершение работы**: Алгоритм завершает работу, когда очередь пуста.

**Характеристики алгоритма**:

* **Временная сложность**: O(V + E), где V — количество вершин, а E — количество рёбер.
* **Пространственная сложность**: O(V + E), так как необходимо хранить граф и очередь.
* **Применимость**: Алгоритм работает быстро для графов с небольшими и средними размерами.

**3. Сравнительный анализ с аналогичными алгоритмами**

| **Критерий** | **Алгоритм BFS** | **Алгоритм Дейкстры** |
| --- | --- | --- |
| Временная сложность | O(V + E) | O(V^2) (с матрицей смежности) или O((V + E) log V) (с использованием кучи) |
| Тип графа | Не взвешенные графы или графы с единичными рёбрами | Взвешенные графы |
| Простота реализации | Простая | Сложная |
| Применимость | Графы с единичными рёбрами или не взвешенные | Графы с положительными весами рёбер |

**Вывод**: Алгоритм BFS является оптимальным для поиска кратчайших путей в графах с единичными рёбрами, так как имеет линейную сложность O(V + E). Алгоритм Дейкстры применим в случаях, когда рёбра графа могут иметь различные веса.

**4. Перечень инструментов, используемых для реализации**

Для реализации алгоритма BFS были использованы следующие инструменты:

* **Язык программирования**: Python 3.9
  + Простой и понятный синтаксис для быстрой реализации.
* **Среда разработки**: Visual Studio Code
  + Поддержка Python, встроенные инструменты для отладки.
* **Модуль для тестирования**: pytest
  + Для автоматизации тестирования алгоритма и проверки корректности работы.

**5. Описание реализации и процесса тестирования**

**Реализация алгоритма**:

Код алгоритма Алгоритм построения кратчайших путей на сети с единичными длинами. был реализован в файле shortest\_path.py. Основные компоненты:

1. **Функция bfs\_shortest\_paths**:
   * Принимает граф в виде словаря смежности и начальную вершину.
   * Возвращает словарь с кратчайшими расстояниями от начальной вершины до всех остальных.
2. **Тестирование**:
   * Для проверки корректности работы алгоритма были написаны тесты с использованием фреймворка pytest.

**Пример входных данных**:

graph\_data = {

0: [1, 2],

1: [0, 3, 4],

2: [0, 4],

3: [1],

4: [1, 2]

}

start\_vertex = 0

**Пример вывода**:

{0: 0, 1: 1, 2: 1, 3: 2, 4: 2}

**Процесс тестирования**:

Для проверки корректности работы алгоритма были подготовлены несколько тестов:

1. **Тестирование простого графа**: Проверка правильности вычисления кратчайших путей в графе с несколькими рёбрами.
2. **Тестирование изолированных вершин**: Проверка случая, когда вершины не соединены рёбрами.
3. **Тестирование графа с циклами**: Проверка корректности работы алгоритма в графах с циклами.
4. **Тестирование disconnected graphs**: Проверка корректности работы алгоритма для разрозненных графов.

**Код тестирования**:

import pytest

from collections import deque

def bfs\_shortest\_paths(graph\_data, start):

distances = {vertex: float('inf') for vertex in graph\_data}

distances[start] = 0

queue = deque([start])

while queue:

current = queue.popleft()

for neighbor in graph\_data[current]:

if distances[neighbor] == float('inf'):

distances[neighbor] = distances[current] + 1

queue.append(neighbor)

return distances

def test\_bfs\_shortest\_paths():

graph\_data = {

0: [1, 2],

1: [0, 3, 4],

2: [0, 4],

3: [1],

4: [1, 2]

}

start\_vertex = 0

result = bfs\_shortest\_paths(graph\_data, start\_vertex)

expected\_result = {0: 0, 1: 1, 2: 1, 3: 2, 4: 2}

assert result == expected\_result

**Преимущества реализации**:

* **Python**: Быстрая разработка, удобство отладки и тестирования.
* **Использование BFS**: Эффективен для графов с единичными рёбрами или не взвешенными графами.

**6. Заключение**

Алгоритм поиска в ширину (BFS) эффективно решает задачу поиска кратчайших путей в графах с единичными рёбрами. Алгоритм обладает линейной временной сложностью O(V + E), что делает его очень быстрым для графов среднего размера. Реализация на Python позволяет быстро разрабатывать и тестировать алгоритм, а также легко адаптировать его для различных задач.

Основные выводы:

1. Алгоритм BFS идеально подходит для графов с единичными рёбрами и не взвешенными графами.
2. **Простота реализации и тестирования**: Реализация алгоритма на Python делает его удобным для быстрого прототипирования, разработки и тестирования. Использование стандартных библиотек, таких как collections.deque, для очередей, значительно ускоряет процесс реализации и упрощает работу с данными.
3. **Подходит для широкого круга задач**: Алгоритм BFS является универсальным для задач, таких как поиск кратчайшего пути в лабиринтах, социальных сетях и других графах, где рёбра имеют одинаковый вес. Он также может быть использован для поиска наименьшего пути до всех вершин графа, что применимо в задачах, связанных с распространением информации или анализом связности.
4. **Преимущества по сравнению с другими алгоритмами поиска кратчайших путей**: В отличие от более сложных алгоритмов, таких как алгоритм Дейкстры, который требует работы с различными весами рёбер, алгоритм BFS значительно быстрее и проще в реализации, когда все рёбра одинаковы или имеют одинаковый вес.

### Перспективы использования

1. **Оптимизация для более сложных задач**: Несмотря на свою простоту, алгоритм BFS может быть использован как основа для более сложных алгоритмов, например, в задачах, где требуется найти кратчайшие пути с ограничениями на количество рёбер или другие условия.
2. **Применение в больших графах**: Для очень больших графов (например, в социальных сетях или в сети интернета вещей) алгоритм BFS остаётся эффективным благодаря своей линейной сложности относительно количества рёбер и вершин.
3. **Использование в реальных приложениях**: Этот алгоритм можно применить в таких областях, как:
   * Навигационные системы, где нужно вычислить кратчайший путь в карте.
   * Сети доставки, где важно быстро находить наименьшие пути между складами и клиентами.
   * Программы для поиска в социальных сетях или анализ их структуры.

### Преимущества реализации на Python

* **Быстрая разработка**: В Python реализовать алгоритм можно быстро, благодаря простоте синтаксиса и наличию мощных стандартных библиотек.
* **Легкость отладки и тестирования**: Python имеет хорошие средства для тестирования, такие как pytest, которые позволяют легко проверять корректность реализации.
* **Гибкость**: Алгоритм можно легко адаптировать под различные требования, например, использовать различные структуры данных для представления графов (списки смежности, матрицы смежности и т.д.).

### Недостатки реализации на Python

* **Скорость выполнения**: Для очень больших графов Python может быть менее эффективен по сравнению с языками, такими как C++, особенно когда речь идёт о графах с миллионами рёбер и вершин. Однако для большинства задач, связанных с графами среднего размера, Python остаётся оптимальным выбором.
* **Меньшая производительность**: Если необходимо обрабатывать огромные графы в реальном времени, лучше использовать более производительные языки, такие как C++ или Java, которые могут обрабатывать такие задачи быстрее, благодаря лучшей оптимизации выполнения.

### Заключение

Алгоритм поиска в ширину (BFS) — это простой, но мощный инструмент для решения задач поиска кратчайших путей в графах с единичными рёбрами. Его эффективность, простота реализации и универсальность делают его отличным выбором для многих приложений, требующих быстрого поиска путей.

Реализация алгоритма на Python позволяет быстро создавать решения и проводить тестирование, а также помогает в решении множества практических задач, таких как анализ графов, поиск кратчайших путей в навигационных системах, социальных сетях и других областях.