МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский технологический университет «МИСИС»

Институт Компьютерных Наук

Отчет

Алгоритм Беллмана-Форда построения кратчайших расстояний.

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

Задача:
Построение кратчайших путей от одной вершины (источника) до всех остальных вершин графа с использованием алгоритма Беллмана-Форда.

Входные данные:

- Ориентированный граф G = (V, E), где:
  - V — множество вершин;
  - E — множество рёбер, каждое с весом w(u, v) для ребра (u, v), где w(u, v) может быть как положительным, так и отрицательным.
- Источник s ∈ V — вершина, от которой необходимо вычислить кратчайшие расстояния.

Выходные данные:

- Массив расстояний от вершины s до всех других вершин графа, либо сообщение о наличии отрицательного цикла, если он существует.

---

2. Теоретическое описание алгоритма

Алгоритм Беллмана-Форда: Алгоритм позволяет найти кратчайшие пути в графе с возможными отрицательными весами рёбер. Основной принцип алгоритма состоит в том, чтобы поочерёдно "освежать" расстояния до всех вершин, проводя несколько итераций по всем рёбрам графа.

Алгоритм состоит из следующих шагов:

1. Инициализация:
   - Задать расстояния до всех вершин как бесконечность, кроме вершины-источника, для которой расстояние равно 0.
2. Основной цикл:
   - Повторить |V| - 1 раз:
     - Для каждого ребра (u, v) обновить расстояние до вершины v, если путь через u даёт более короткое расстояние.
3. Проверка на отрицательные циклы:

- Пройти по всем рёбрам ещё раз. Если на какой-то вершине можно уменьшить расстояние, значит, граф содержит отрицательный цикл.

Характеристики алгоритма:

- Временная сложность: O(V×E), где V — количество вершин, E — количество рёбер.
- Пространственная сложность: O(V), так как храним массив кратчайших расстояний.

---

3. Перечень инструментов, использованных для реализации

Для реализации алгоритма были использованы следующие инструменты:

- **Язык программирования**: Python 3.9+
- **Среда разработки**: Visual Studio Code, PyCharm
- **Тестирование**: Фреймворк `pytest` для написания и выполнения юнит-тестов.
- **Система контроля версий**: Git

---

4. Описание реализации и процесса тестирования

Реализация алгоритма: Алгоритм был реализован с использованием класса `Graph`, который поддерживает методы для добавления рёбер и выполнения алгоритма Беллмана-Форда.

- Метод `add_edge(u, v, weight)` добавляет ребро с весом weight между вершинами u и v.
- Метод `bellman_ford(start)` выполняет сам алгоритм, начиная с вершины `start` и возвращает массив расстояний от источника до всех остальных вершин.

Процесс тестирования: Для проверки корректности работы алгоритма был написан набор тестов с использованием фреймворка `pytest`. Тесты охватывают различные сценарии:

1. Пустой граф.
2. Граф с одним ребром.
3. Граф с положительными и отрицательными весами.
4. Граф с отрицательным циклом.
5. Разрежённый граф с не связанными вершинами.

Тесты автоматически проверяют, корректно ли вычисляются кратчайшие расстояния и правильно ли обрабатываются отрицательные циклы.

Код тестов:

```
import pytest
from bellman_ford import Graph
```

```
def test_empty_graph():
    g = Graph(3)
    distances = g.bellman_ford(0)
    assert distances == [0, float("inf"), float("inf")]

def test_single_edge():
    g = Graph(2)
    g.add_edge(0, 1, 5)
    distances = g.bellman_ford(0)
    assert distances == [0, 5]

def test_graph_with_positive_and_negative_weights():
    g = Graph(5)
    g.add_edge(0, 1, -1)
    g.add_edge(0, 2, 4)
    g.add_edge(1, 2, 3)
    g.add_edge(1, 3, 2)
    g.add_edge(1, 4, 2)
    g.add_edge(3, 2, 5)
    g.add_edge(3, 1, 1)
    g.add_edge(4, 3, -3)
    distances = g.bellman_ford(0)
    assert distances == [0, -1, 2, -2, 1]

def test_negative_cycle():
    g = Graph(3)
    g.add_edge(0, 1, 1)
    g.add_edge(1, 2, -1)
    g.add_edge(2, 0, -1)
    distances = g.bellman_ford(0)
    assert distances is None

def test_disconnected_graph():
    g = Graph(4)
    g.add_edge(0, 1, 2)
    g.add_edge(2, 3, 3)
    distances = g.bellman_ford(0)
    assert distances == [0, 2, float("inf"), float("inf")]
```

---

5. Пример входных и выходных данных

Пример входных данных:

```
5 8
0 1 -1
0 2 4
1 2 3
1 3 2
1 4 2
3 2 5
3 1 1
```

```
4 3 -3
0
```

Пример вывода программы:

```
Кратчайшие расстояния от источника 0:
Вершина 0: 0
Вершина 1: -1
Вершина 2: 2
Вершина 3: -2
Вершина 4: 1
```

---

6. Заключение

Алгоритм Беллмана-Форда эффективен для нахождения кратчайших путей в графах с возможными отрицательными весами рёбер. Реализация на Python с использованием фреймворка `pytest` для тестирования позволяет быстро разрабатывать и проверять алгоритм.

Основные выводы:

1. Алгоритм корректно работает для графов с отрицательными рёбрами, вычисляя кратчайшие расстояния.
2. Реализация корректно обрабатывает случай наличия отрицательных циклов.
3. Процесс тестирования с использованием `pytest` позволяет легко автоматизировать проверку правильности работы алгоритма.