Сетевое программирование

Лекция 3

Структуры данных в библиотеке NetworkX. Классы Graph, DiGraph, MultiGraph, MultiDiGraph

Усіпов Н.М., PhD · КазНУ имени аль-Фараби

Цель и план лекции

Цель

Изучение внутреннего устройства графов в библиотеке NetworkX, назначения четырёх основных классов и работы с атрибутами узлов и рёбер, что необходимо для осознанного выбора структуры данных при моделировании сети.

1. Введение
2. Четыре основных класса графов
3. Просмотр содержимого графа
4. Выбор структуры данных
5. Вычислительная сложность основных операций
6. Атрибуты графа, узлов и рёбер
7. Преобразование мультиграфа в простой граф
8. Устройство внутренних структур

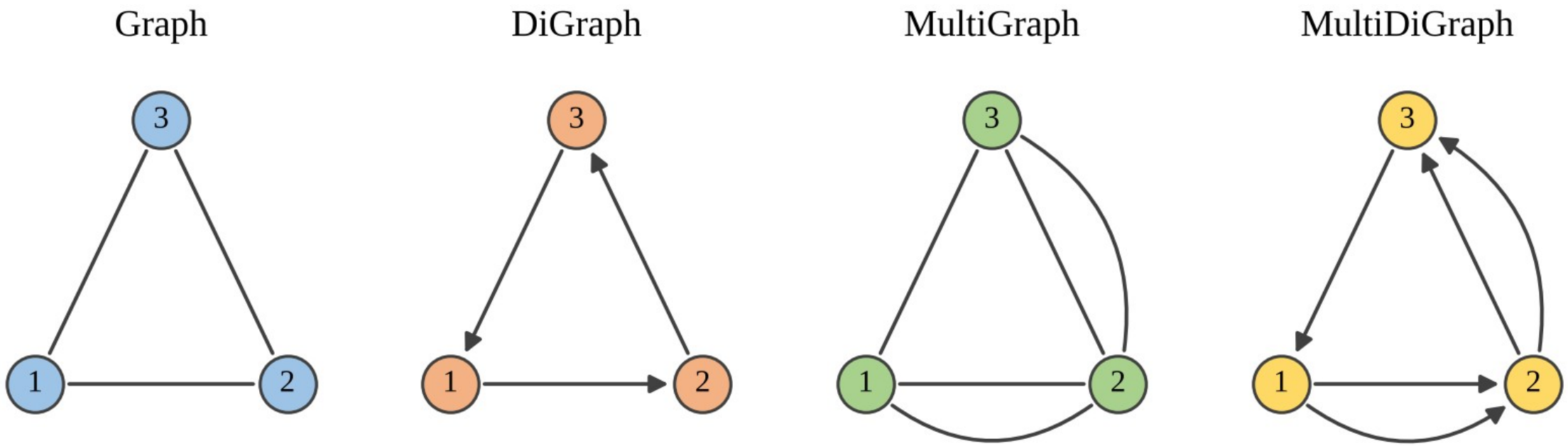
1 Введение

- Эффективность сетевой программы во многом определяется выбранной структурой данных.
- Графы в NetworkX организованы по принципу «словарь словарей словарей». Внешний словарь сопоставляет каждому узлу его соседей, вложенный словарь сопоставляет соседу данные о ребре, а внутренний содержит атрибуты этого ребра.

2 Четыре основных класса графов

- Библиотека предоставляет четыре класса, различающихся направленностью рёбер и возможностью хранить параллельные рёбра.

2 Четыре основных класса графов (продолжение)



Четыре основных класса графов NetworkX

2 Четыре основных класса графов (продолжение)

Graph

Неориентированный граф без кратных рёбер. Подходит для моделирования сети с симметричными каналами, когда между двумя устройствами учитывается не более одного соединения.

DiGraph

Ориентированный граф. Для каждого узла отдельно хранятся словари исходящих дуг (`succ`) и входящих дуг (`pred`), что ускоряет доступ к предшественникам, но требует больше памяти.

MultiGraph

Неориентированный граф с кратными рёбрами. Каждому ребру между парой узлов присваивается ключ.

MultiDiGraph

Ориентированный граф с кратными рёбрами. Он описывает, например, несколько однонаправленных каналов между устройствами или несколько различных потоков данных.

2 Четыре основных класса графов (продолжение)

- Следует учитывать, что не все алгоритмы определены для каждого класса. Многие алгоритмы работают только с простыми графами, поэтому мультиграф перед анализом иногда приходится преобразовывать, заменяя группу параллельных рёбер одним ребром с суммарной пропускной способностью или минимальной задержкой.

3 Просмотр содержимого графа

- Для обхода графа предусмотрены представления `G.nodes`, `G.edges`, `G.adj` и `G.degree`.
- Для отбора элементов удобно использовать генераторы списков. Например, выражение, перебирающее `G.edges(data="delay")`, позволяет выделить каналы с задержкой больше заданного порога.
- Следует избегать изменения структуры графа непосредственно в цикле обхода его узлов или рёбер: это приведёт к ошибке, так как размер словаря изменится во время итерации.

4 Выбор структуры данных

- Каналы симметричны, параллельные линии не важны – `Graph`.
- Важно направление передачи – `DiGraph`.
- Существуют параллельные каналы с разными параметрами – `MultiGraph`.
- Важны и направление, и параллельные каналы – `MultiDiGraph`.
- Выбирать следует самый простой класс, который достаточно точно описывает задачу.

5 Вычислительная сложность основных операций

- Благодаря использованию хеш-таблиц основные операции выполняются в среднем за постоянное время независимо от размера графа.
- Для задач, требующих интенсивных численных вычислений, представление NetworkX не является наиболее компактным.

6 Атрибуты графа, узлов и рёбер

- В NetworkX атрибуты хранятся на трёх уровнях. Атрибуты всего графа находятся в словаре `G.graph`, атрибуты узлов – в представлении `G.nodes`, атрибуты рёбер – в представлении `G.edges`.
- Основные методы изменения графа:
- `add_node`, `add_nodes_from` – добавление узлов;
- `add_edge`, `add_edges_from`, `add_weighted_edges_from` – добавление рёбер;
- `remove_node`, `remove_edge` – удаление элементов;
- `set_node_attributes`, `set_edge_attributes` – массовое задание атрибутов.
- Если при добавлении ребра какой-либо из его узлов отсутствует, узел создаётся автоматически.

6 Атрибуты графа, узлов и рёбер (продолжение)

- Важно различать копию и представление. Методы `subgraph()` и `edge_subgraph()` возвращают представление (view): данные не копируются, а изменения атрибутов отражаются на исходном графе.

6 Атрибуты и параллельные каналы

Листинг 3.1.

```
import networkx as nx

G = nx.Graph(name="Опорная сеть")
G.add_node("R1", vendor="Eltex", role="core")
G.add_edge("R1", "R2", bandwidth=10000, delay=2)
G["R1"]["R2"]["delay"] = 3            # изменение атрибута

print(G.graph["name"], G.nodes["R1"]["role"])
print("Структура adj:", dict(G.adj["R1"]))

M = nx.MultiGraph()
M.add_edge("R1", "R2", key="fiber1", bandwidth=10000)
M.add_edge("R1", "R2", key="fiber2", bandwidth=40000)
print("Рёбер R1-R2:", M.number_of_edges("R1", "R2"))
total = sum(d["bandwidth"] for d in M["R1"]["R2"].values())
print("Суммарная полоса:", total, "Мбит/с")

D = nx.DiGraph()
D.add_edge("A", "B", bandwidth=100)
print("A->B:", D.has_edge("A", "B"), "B->A:", D.has_edge("B", "A"))
```

6 Результат: Атрибуты и параллельные каналы

```
Опорная сеть core
Структура adj: {'R2': {'bandwidth': 10000, 'delay': 3}}
Рёбер R1-R2: 2
Суммарная полоса: 50000 Мбит/с
A->B: True B->A: False
```

Из приведённого примера видно, что словарь смежности узла R1 содержит соседа R2 и атрибуты соединяющего их ребра.

7 Преобразование мультиграфа в простой граф

- Для вычисления многих метрик мультиграф приходится сводить к простому графу.

7 Агрегация параллельных каналов

Листинг 3.2.

```
import networkx as nx

M = nx.MultiGraph()
M.add_edge("A", "B", bw=10, delay=5)
M.add_edge("A", "B", bw=40, delay=2)
M.add_edge("B", "C", bw=10, delay=4)

S = nx.Graph()
for u, v, d in M.edges(data=True):
    if S.has_edge(u, v):
        S[u][v]["bw"] += d["bw"]
        S[u][v]["delay"] = min(S[u][v]["delay"], d["delay"])
    else:
        S.add_edge(u, v, bw=d["bw"], delay=d["delay"])

for u, v, d in S.edges(data=True):
    print(u, v, d)
slow = [(u, v) for u, v, b in S.edges(data="bw") if b < 20]
print("Каналы менее 20 Гбит/с:", slow)
```

7 Результат: Агрегация параллельных каналов

```
A B {'bw': 50, 'delay': 2}
B C {'bw': 10, 'delay': 4}
Каналы менее 20 Гбит/с: [('B', 'C')]
```

Таким образом, выбор класса графа и способа агрегации являются частью постановки задачи и должны быть обоснованы в отчёте по работе.

8 Устройство внутренних структур

- Рассмотрим устройство объекта `nx.Graph` подробнее. Основными его полями являются `_node` – словарь атрибутов узлов, `_adj` – словарь смежности и `graph` – словарь атрибутов графа.
- Такая организация определяет важные следствия. Узлом может служить любой хешируемый объект, но изменяемый список им быть не может, так как значение хеша не должно меняться.
- Значение `None` не может использоваться в качестве узла, поскольку оно играет служебную роль в функциях библиотеки.

9 Взаимодействие с другими структурами данных

На практике графы приходится преобразовывать в другие представления.

Словарь списков и словарь словарей

Функции `to_dict_of_lists` и `to_dict_of_dicts` позволяют получить обычные структуры Python, удобные для сериализации.

Разреженные матрицы

Функция `to_scipy_sparse_array` создаёт разреженную матрицу, что полезно для спектрального анализа больших сетей.

Таблицы

Функция `to_pandas_edgelist` формирует таблицу рёбер для статистической обработки атрибутов.

Формат JSON

Модуль `networkx.readwrite.json_graph` подготавливает граф для передачи в веб-приложения и средства визуализации.

9 Взаимодействие с другими структурами данных (продолжение)

- Для экономии памяти при работе с атрибутами стоит хранить в них только простые значения.
- Подграфы, создаваемые как представления, позволяют эффективно выполнять анализ сценариев.
- Важным инструментом разработки также являются модульные тесты. Например, для функции, вычисляющей суммарную полосу параллельных каналов, следует подготовить небольшой мультиграф с заранее известным ответом.

10 Разбор примера

1. Пусть маршрутизаторы A и B соединены тремя параллельными линиями с пропускной способностью 10, 40 и 100 Гбит/с и задержками 5, 2 и 3 мс соответственно.
2. Модель `MultiGraph` хранит для этой пары три ребра с ключами.
3. Для задачи оценки полосы агрегированный канал имеет ёмкость 10 + 40 + 100 = 150 Гбит/с.
4. Для задачи поиска наименьшей задержки следует взять минимальное значение – 2 мс.
5. Если учитывать только соединения не менее 40 Гбит/с, суммарная ёмкость равна 140 Гбит/с, а минимальная задержка – по-прежнему 2 мс.
6. Рассмотрим запись `G.add_edge("A", "B", bw=10)`, выполненную дважды с разными значениями атрибута в `Graph`.
7. Число рёбер останется равным одному, а значение атрибута будет заменено последним. В классе `MultiGraph` та же последовательность команд создаст два ребра.

11 Вопросы для обсуждения на занятии

Выбор идентификатора

Что лучше использовать как идентификатор узла: IP-адрес, имя устройства или внутренний числовой номер?

Параллельные каналы

Как объединить параллельные каналы, если часть из них находится в резерве?

Ошибки

Какие ошибки могут появиться из-за того, что атрибут с опечаткой в имени незаметно заменяется единицей?

Контрольные вопросы

1. Почему в NetworkX используются словари Python?
2. Опишите принцип организации данных «словарь словарей словарей».
3. Назовите четыре основных класса графов и укажите их различия.
4. Что произойдёт при повторном добавлении существующего ребра в объект класса Graph?
5. В каких случаях сети нужно моделировать мультиграфами? Приведите примеры.
6. На каких трёх уровнях в NetworkX можно хранить атрибуты?
7. Чем представление подграфа отличается от копии? Когда необходимо использовать копию?
8. Как преобразовать мультиграф к простому графу для вычисления кратчайшего пути?

Список литературы

1. Hagberg A., Schult D., Swart P. Exploring network structure, dynamics, and function using NetworkX // Proc. 7th Python in Science Conference (SciPy 2008). – Pasadena, 2008. – P. 11–15.
2. NetworkX Documentation [Электронный ресурс]. – URL: https://networkx.org/documentation/stable/
3. Newman M. Networks. – 2nd ed. – Oxford: Oxford University Press, 2018. – 780 p.
4. Cormen T., Leiserson C., Rivest R., Stein C. Introduction to Algorithms. – 4th ed. – Cambridge, MA: MIT Press, 2022.
5. Python Documentation [Электронный ресурс]. – URL: https://docs.python.org/3/