АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ЛЕКЦИЯ 6

Поиск в ширину. Знакомство с графами

Очередь открывает вершины по слоям расстояния от старта.

06

ИЗ 15 ЛЕКЦИЙ



МОДУЛЬ 2 Хеш-таблицы. Графы

1 КУРС / 2 АКАДЕМИЧЕСКИХ ЧАСА / PYTHON 3

Цель и содержание лекции

ЦЕЛЬ И РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

представлять связи между объектами в виде графа, выполнять поиск в ширину с помощью очереди и восстанавливать кратчайший по числу рёбер путь между двумя вершинами.

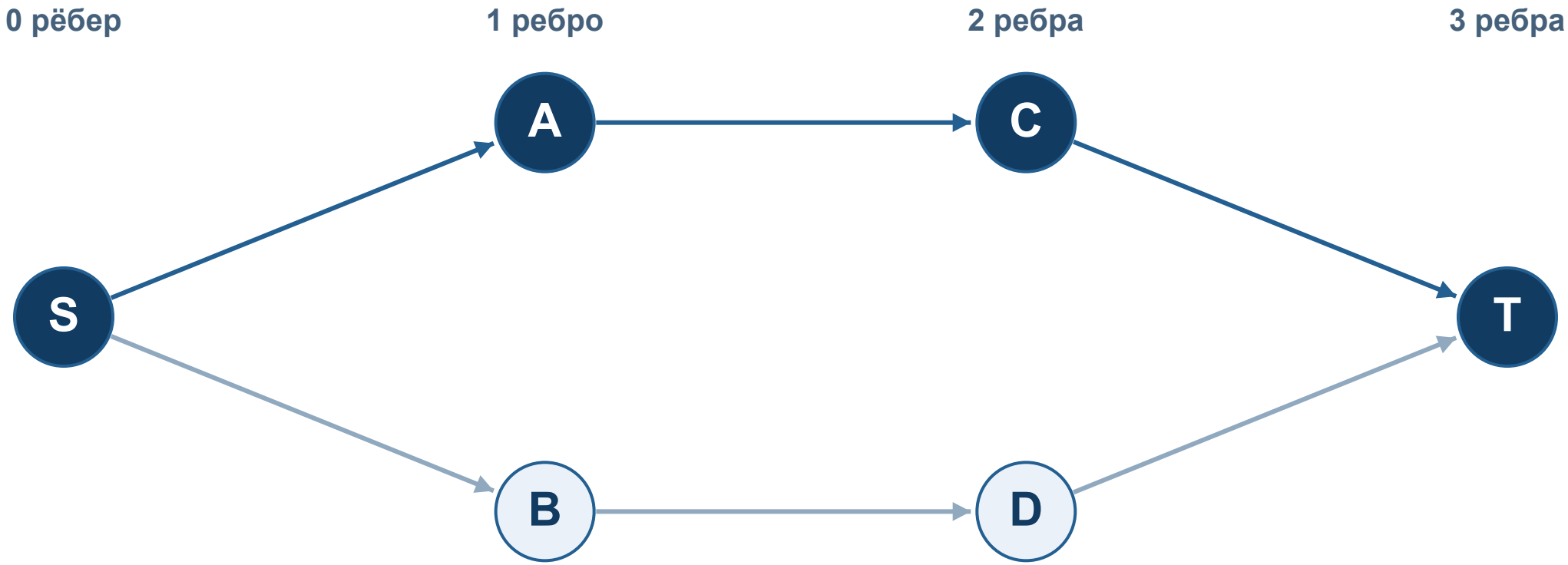
- Описывать граф списком смежности.
- Трассировать BFS с очередью и отметками посещения.
- Восстанавливать путь с минимальным числом рёбер.

ПЛАН ЛЕКЦИИ

1. **Вершины, рёбра и список смежности**
2. **Уровни поиска и очередь FIFO**
3. **Трассировка поиска по сети знакомств**
4. **Восстановление пути на Python**
5. **Обоснование корректности и оценка сложности**

Граф открывается слой за слоем

СХЕМА



Есть два пути из S в T длины 3. Порядок соседей определяет, какой из них вернёт реализация.

Вершины, рёбра и список смежности

01 ТЕМА

Когда задача описывает связи между объектами, удобно выделить сами объекты и допустимые переходы между ними. Например, вершина может обозначать станцию метро, а ребро — прямой переезд между станциями. В сети знакомств вершина соответствует человеку, а ребро может означать, что один человек знает другого. Такая модель называется **графом**. Число вершин будем обозначать `V`, число рёбер — `E`.

У **ориентированного** ребра есть направление. Если существует ребро `a → b`, переход от `a` к `b` разрешён, а допустимость обратного перехода проверяется отдельно. В **неориентированном** графе ребро позволяет двигаться в обоих направлениях. Поэтому в списке смежности неориентированное ребро записывают дважды, по одной записи у каждого конца. В графе могут встречаться циклы, то есть маршруты, возвращающие нас к исходной вершине, и изолированные вершины без рёбер.

Вершины, рёбра и список смежности

01 ПРОДОЛЖЕНИЕ

Список смежности хранит для каждой вершины список её соседей. В Python вершины удобно обозначать строками, а списки соседей хранить в словаре. Вершина с пустым списком соседей остаётся частью графа. Можно также использовать матрицу смежности, где пара индексов указывает на наличие ребра. Она занимает $O(V^2)$ памяти, а списки смежности занимают $O(V + E)$. При малом числе связей на вершину списки позволяют обходить только имеющиеся рёбра.

Уровни поиска и очередь FIFO

02 ТЕМА

Поиск в ширину, или BFS от английского breadth-first search, отвечает на два связанных вопроса. Можно ли добраться до цели из заданной вершины? Если можно, сколько рёбер содержит кратчайший путь? Каждый переход в этой постановке имеет одинаковую стоимость. Например, если важна минимальная сумма времени в минутах, а поездки по разным рёбрам занимают разное время, потребуется алгоритм для взвешенного графа.

Начальная вершина находится на уровне 0. Её непосредственные соседи составляют уровень 1. Ещё не открытые соседи уровня 1 образуют уровень 2. Поиск продолжает раскрывать уровни, пока не встретит цель или пока не исчерпает доступные вершины. Такое движение по уровням реализует **очередь FIFO**. Первым обрабатывается элемент, добавленный раньше остальных. В `collections.deque` операции `append` и `popleft` работают за `O(1)`. У списка Python `pop(0)` занимает `O(n)`, поскольку сдвигает оставшиеся элементы.

Уровни поиска и очередь FIFO

02 ПРОДОЛЖЕНИЕ

Мы помечаем вершину открытой в момент постановки в очередь. Допустим, два человека одного уровня знают Пегги. После добавления Пегги первым из них второй уже увидит отметку и пропустит её. Та же отметка защищает от повторных обходов при наличии циклов. В программе отметкой будет запись в словаре предшественников. Она одновременно хранит вершину, из которой мы впервые пришли к соседу.

Трассировка поиска по сети знакомств

03 ТЕМА

Рассмотрим сеть из главы 6 книги А. Бхаргавы. У пользователя есть знакомые Алиса, Боб и Клэр. Алиса знает Пегги. Боб знает Ануджа и Пегги. Клэр знает Тома и Джонни. Ищем ближайшего продавца манго. В учебной программе роль продавца закрепим за Томом, а знакомых будем добавлять в очередь в порядке их записи в словаре.

Трассировка поиска по сети знакомств

03 ПРОДОЛЖЕНИЕ

Извлечённая вершина	Очередь после добавления новых соседей
Старт	Алиса, Боб, Клэр
Алиса	Боб, Клэр, Пегги
Боб	Клэр, Пегги, Анудж
Клэр	Пегги, Анудж, Том, Джонни
Пегги	Анудж, Том, Джонни
Анудж	Том, Джонни
Том	Поиск завершён

Трассировка поиска по сети знакомств

03 ПРОДОЛЖЕНИЕ

У Тома предшественник Клэр, а у Клэр — стартовая вершина. Читая цепочку предшественников от цели к началу, получаем обратный порядок. После разворота маршрут проходит от пользователя к Клэр, а затем к Тому. В нём три вершины и два ребра. Если бы продавцами были и Пегги, и Том, поиск в указанном порядке вернул бы Пегги. Обе вершины находятся на уровне 2, поэтому найденное расстояние осталось бы тем же.

Восстановление пути на Python

04 ТЕМА

Функция принимает список смежности, начальную вершину и цель. Она возвращает список вершин пути или `None`, если цель недостижима. Вершины обозначены строками. Входной словарь содержит ключ для каждой вершины, включая вершины без исходящих рёбер. Чтобы найти ближайшую вершину с некоторым свойством, вместо проверки равенства цели можно проверять свойство извлечённой вершины.

Восстановление пути на Python

04 ПРИМЕР PYTHON

06_01.py / фрагмент 1 из 2

```
from collections import deque


def shortest_path(graph, start, goal):
    queue = deque([start])
    parent = {start: None}

    while queue:
        current = queue.popleft()
        if current == goal:
            path = []
            while current is not None:
                path.append(current)
                current = parent[current]
            return path[::-1]

        for neighbor in graph[current]:
            if neighbor not in parent:
                parent[neighbor] = current
                queue.append(neighbor)
```

Восстановление пути на Python

06_01.py / фрагмент 2 из 2

```
    return None


graph = {
    'вы': ['Алиса', 'Боб', 'Клэр'],
    'Алиса': ['Пегги'],
    'Боб': ['Анудж', 'Пегги'],
    'Клэр': ['Том', 'Джонни'],
    'Пегги': [],
    'Анудж': [],
    'Том': [],
    'Джонни': [],
    'Дана': [],
}

path = shortest_path(graph, 'вы', 'Том')
print(' → '.join(path))
print(len(path) - 1)
print(shortest_path(graph, 'вы', 'Дана'))
print(shortest_path(graph, 'вы', 'вы'))
```

РЕЗУЛЬТАТ ЗАПУСКА

```
вы → Клэр → Том
2
None
['вы']
```

Восстановление пути на Python

04 ПРОДОЛЖЕНИЕ

Последняя проверка показывает особый случай. Когда начальная вершина совпадает с целью, путь состоит из одной вершины, а его длина равна нулю. Поиск из Тома к Клэр на тех же данных вернёт `None`. Список смежности задаёт направление связей, и ребро от Клэр к Тому обратного перехода не добавляет.

Обоснование корректности и оценка сложности

В начале очередь содержит только старт, для которого кратчайшее расстояние известно и равно нулю. Когда обрабатывается вершина на уровне `d`, любой её новый сосед получает маршрут длины `d + 1`. Предположим, что к соседу есть путь короче. Тогда предыдущая вершина на этом коротком пути принадлежит более раннему уровню и была обработана раньше. Она уже должна была открыть соседа, что противоречит отсутствию отметки. Поэтому первое открытие фиксирует кратчайшее расстояние. Цепочка предшественников проходит по уровням в обратном порядке и даёт путь этой длины.

Если очередь опустела, все достижимые вершины были обработаны. Иначе можно было бы взять путь к необработанной достижимой вершине и найти на нём первую неоткрытую вершину. Её предшественник на пути уже рассмотрен и добавил бы её в очередь. Этот аргумент объясняет, почему `None` означает отсутствие пути в заданном направлении.

Обоснование корректности и оценка сложности

05 ПРОДОЛЖЕНИЕ

Каждая вершина попадает в очередь не более одного раза, а каждый список соседей просматривается не более одного раза. Для списков смежности время обхода в худшем случае составляет `O(V + E)`, дополнительная память на очередь и предшественников — `O(V)`. Для неориентированного графа каждое ребро рассматривается с двух концов, что меняет постоянный множитель. При ранней остановке работа ограничивается просмотренной частью графа. Восстановление пути требует времени, пропорционального числу вершин в нём.

Что нужно уметь после лекции

ИТОГИ

Направление рёбер и список смежности определяют возможные переходы. Очередь удерживает порядок обработки по расстоянию от старта. Словарь предшественников позволяет одновременно исключить повторные посещения и восстановить кратчайший путь.

> Очередь открывает вершины по слоям расстояния от старта.

Проверьте понимание

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как в списке смежности представлены ориентированное ребро и изолированная вершина?
2. В какой момент нужно пометить вершину открытой и как это повлияет на очередь в примере с Пегги?
3. Почему вершина, открытая на уровне 2, не может иметь прямого ребра от старта?
4. Как отличаются число вершин в пути и его длина?
5. Что произойдёт с порядком обхода, если использовать стек?

Самостоятельная работа

ПРАКТИКА

1. Запишите состояния очереди для сети знакомств, если Боб указан перед Алисой. Сравните предшественников Пегги в обоих порядках.
2. Добавьте ребро от Джонни к стартовой вершине. Проследите полный обход при поиске Даны и объясните, почему цикл не приводит к зацикливанию программы.
3. Напишите функцию, которая возвращает словарь расстояний от старта до всех достижимых вершин. Проверьте нулевое расстояние до старта и отсутствие Даны в словаре.

Источники и документация

ЧТЕНИЕ

ОСНОВНОЕ ЧТЕНИЕ

А. Бхаргава. Грокаем алгоритмы

СПб.: Питер, 2017. Глава 6. Поиск в ширину. С. 127–150.

> При чтении сначала вручную пройдите один пример, затем запустите код и сравните результат.