АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ЛЕКЦИЯ 7

Алгоритм Дейкстры

07

ИЗ 15 ЛЕКЦИЙ

При неотрицательных весах ближайшую вершину можно зафиксировать.



МОДУЛЬ 2 Хеш-таблицы. Графы

1 КУРС / 2 АКАДЕМИЧЕСКИХ ЧАСА / PYTHON 3

Цель и содержание лекции

ЦЕЛЬ И РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

находить кратчайшие пути от заданной вершины во взвешенном графе с неотрицательными весами, объяснять роль релаксации и восстанавливать маршрут по предшественникам.

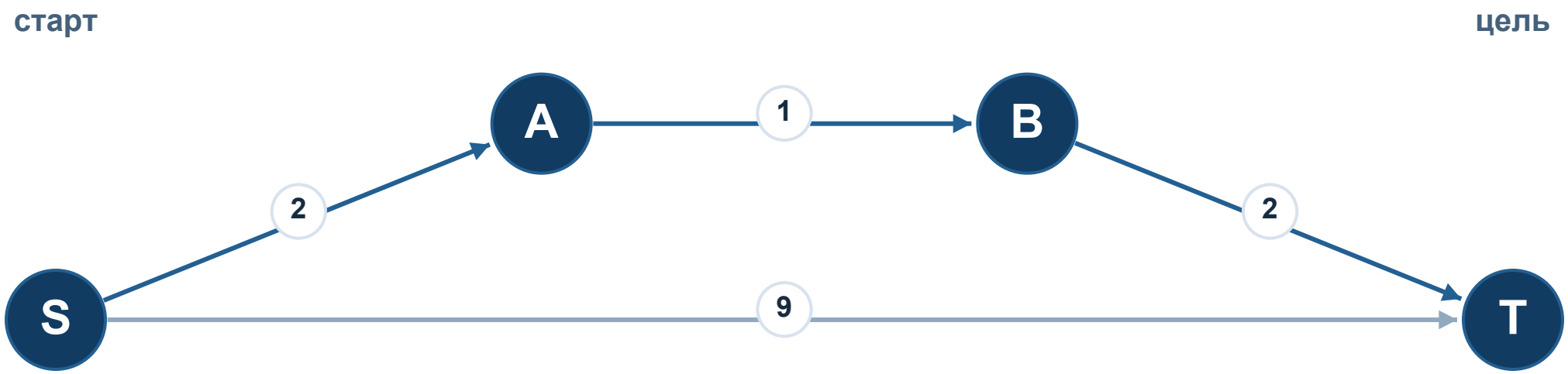
- Различать число рёбер и суммарный вес маршрута.
- Вручную обновлять расстояния при релаксации.
- Объяснять роль неотрицательных весов и очереди с приоритетом.

ПЛАН ЛЕКЦИИ

1. **Веса рёбер и стоимость маршрута**
2. **Предварительные расстояния и релаксация**
3. **Ручная трассировка и восстановление маршрута**
4. **Реализация на Python и оценка сложности**
5. **Обоснование корректности и границы применимости**

Самый лёгкий маршрут может быть длиннее

СХЕМА



Прямое ребро стоит 9. Маршрут S → A → B → T состоит из трёх рёбер, но его стоимость всего 5.

Веса рёбер и стоимость маршрута

01 ТЕМА

Поиск в ширину находит маршрут с минимальным числом рёбер. Во взвешенном графе каждое ребро имеет числовую характеристику, например время поездки. Если маршрут проходит по трём рёбрам с весами 2, 3 и 1, его стоимость равна 6. Такой маршрут может быть быстрее маршрута из двух рёбер с весами 6 и 1. При моделировании важно выбрать общую единицу измерения и убедиться, что стоимости переходов складываются.

Алгоритм Дейкстры вычисляет минимальную сумму весов от одной начальной вершины до всех достижимых вершин. Для задачи с одной целью выполнение можно завершить, когда расстояние до этой цели стало окончательным. Предполагается, что каждый вес — конечное неотрицательное число. Нулевые рёбра допустимы. В ориентированном графе ребро из u в v и обратное ребро задаются отдельно. В неориентированном графе одну связь представляют двумя переходами с одинаковым весом.

Предварительные расстояния и релаксация

02 ТЕМА

Создадим словарь `distance`. Старту присвоим 0, остальным вершинам — бесконечность. Конечная оценка `distance[v]` соответствует стоимости конкретного уже найденного пути к `v`. Она может уменьшиться, если обнаружится более дешёвый путь. Бесконечность означает, что маршрут пока неизвестен. Второй словарь `parent` хранит предшественника для каждого найденного пути.

На каждом шаге выбираем необработанную вершину `u` с наименьшей конечной оценкой. Расстояние до неё становится окончательным. Для каждого исходящего ребра `u → v` веса `w` проверяем стоимость пути через `u`: `distance[u] + w`. Если она меньше `distance[v]`, обновляем расстояние и записываем `parent[v] = u`. Эту операцию называют **релаксацией ребра**. При равной стоимости сохраняем старый маршрут. Поэтому результат при одинаковых стоимостях может зависеть от порядка перебора вершин, а минимальная стоимость остаётся прежней.

Предварительные расстояния и релаксация

02 ПРОДОЛЖЕНИЕ

Заметьте, что при выборе u сравнивают длины маршрутов от старта. Если до одной вершины уже пройдено 20 минут, её ребро весом 1 даёт путь стоимостью 21. Ребро весом 4 из другой вершины, до которой дошли за 5 минут, даёт путь стоимостью 9. Вес последнего перехода сам по себе не определяет, какой маршрут дешевле.

Ручная трассировка и восстановление маршрута

В примере из главы 7 из старта ведут рёбра в A за 6 минут и в B за 2 минуты. Из B можно перейти в A за 3 минуты или сразу к финишу за 5 минут. Из A до финиша остаётся 1 минута. Запишем оценки после обработки очередной вершины.

Обработанная вершина	A	B	Финиш	Изменения предшественников
Старт	6	2	∞	Для A и B записан старт
B	5	2	7	Для A и финиша записана B
A	5	2	6	Для финиша записана A
Финиш	5	2	6	Изменений нет

Ручная трассировка и восстановление маршрута

03 ПРОДОЛЖЕНИЕ

Проверим шаг с вершиной `B`. Путь к `A` через неё стоит `2 + 3 = 5`, поэтому он улучшает оценку 6. Стоимость пути до финиша через `B` равна `2 + 5 = 7`, поэтому бесконечность заменяется семёркой. Следующей окончательной становится оценка 5 для `A`. Ребро от `A` к финишу даёт новое улучшение `5 + 1 = 6`. Первая запись о достижении финиша стоила 7, поэтому ранняя остановка при обнаружении финиша вернула бы завышенную стоимость.

От финиша идём по словарю предшественников к `A`, затем к `B`, затем к старту. Разворачиваем полученный список. Путь `старт → B → A → финиш` содержит три ребра и стоит 6. Оба пути с двумя рёбрами в этом графе стоят по 7 минут.

Ручная трассировка и восстановление маршрута

03 ПРОДОЛЖЕНИЕ

Цепочка обменов

В другом примере книги Рама хочет обменять книгу на пианино с минимальной суммарной доплатой. Вершинами графа здесь служат предметы. Обменять книгу на постер можно бесплатно, на пластинку — за 5 долларов. С постера можно перейти к бас-гитаре за 30 или к барабану за 35. С пластинки эти обмены обходятся соответственно в 15 и 20. Бас-гитару можно обменять на пианино с доплатой 20, барабан — с доплатой 10.

После обработки постера известны маршруты к бас-гитаре за 30 и к барабану за 35. Обработка пластинки улучшает их до 20 и 25. Сначала алгоритм обработает бас-гитару, поэтому первая оценка для пианино равна 40. Затем обработает барабан и уменьшит оценку до 35. Цепочка предшественников даёт маршрут `книга → пластинка → барабан → пианино`. Бесплатный первый обмен допустим, но дальнейшие расходы по этой ветви выше. Дешёвый переход сам по себе не определяет стоимость всего маршрута.

Реализация на Python и оценка сложности

В коде каждой вершине соответствует словарь её соседей и весов переходов. Предполагаем, что вершины обозначены строками, все концы рёбер записаны как ключи внешнего словаря, старт существует, а веса удовлетворяют условию выше. Для учебного примера ищем ближайшую необработанную вершину линейным просмотром словаря.

Реализация на Python и оценка сложности

04 ПРИМЕР PYTHON

07_01.py / фрагмент 1 из 3

```
from math import inf


def dijkstra(graph, start):
    distance = dict.fromkeys(graph, inf)
    distance[start] = 0
    parent = {start: None}
    settled = set()

    while True:
        current = None
        best = inf
        for vertex, cost in distance.items():
            if vertex not in settled and cost < best:
                current, best = vertex, cost
        if current is None:
            return distance, parent
```

Реализация на Python и оценка сложности

07_01.py / фрагмент 2 из 3

```
        settled.add(current)
        for neighbor, weight in graph[current].items():
            candidate = best + weight
            if candidate < distance[neighbor]:
                distance[neighbor] = candidate
                parent[neighbor] = current


def restore_path(parent, goal):
    if goal not in parent:
        return None
    path = []
    while goal is not None:
        path.append(goal)
        goal = parent[goal]
    return path[::-1]
```

Реализация на Python и оценка сложности

04 ПРИМЕР PYTHON

07_01.py / фрагмент 3 из 3

```
graph = {
    'старт': {'A': 6, 'B': 2},
    'A': {'финиш': 1},
    'B': {'A': 3, 'финиш': 5},
    'финиш': {},
    'резерв': {},
}

distance, parent = dijkstra(graph, 'старт')
print(distance['финиш'])
print(' → '.join(restore_path(parent, 'финиш')))
print(distance['резерв'], restore_path(parent, 'резерв'))
```

РЕЗУЛЬТАТ ЗАПУСКА

```
6
старт → B → A → финиш
inf None
```

Реализация на Python и оценка сложности

04 ПРОДОЛЖЕНИЕ

Линейный выбор минимума стоит `O(V)` на шаг, таких шагов не более `V`. Каждое ребро просматривается не более одного раза. Итоговое время — `O(V² + E)`, дополнительная память — `O(V)`. Для разреженного графа можно хранить кандидатов в двоичной куче. В распространённой реализации на `heapq` при улучшении оценки добавляют новую пару «стоимость, вершина», а при извлечении сверяют стоимость с текущей оценкой и пропускают устаревшие пары. При таком подходе в куче может быть `O(E)` записей.

Обоснование корректности и границы применимости

Предположим, что выбрана необработанная вершина u и до неё существует маршрут дешевле текущей оценки. На этом маршруте найдём первую ещё необработанную вершину x. Её предшественник уже обработан, поэтому ребро к x уже релаксировали. Оценка x не выше стоимости префикса маршрута до x. Поскольку оставшиеся рёбра неотрицательны, этот префикс стоит не больше всего предполагаемого маршрута. Следовательно, оценка x была бы меньше оценки u. Это противоречит выбору u как вершины с минимумом. На первом шаге правильность нулевого расстояния до старта следует из тех же неотрицательных весов.

Проверим, где доказательство перестаёт работать. Пусть S → A весит 2, S → B весит 5, а B → A весит −4. Сразу после обработки старта алгоритм выберет A с оценкой 2. Однако маршрут S → B → A имеет стоимость 1. Ошибка возможна даже в графе без циклов. При наличии отрицательных рёбер применяют алгоритм Беллмана — Форда, который также позволяет обнаруживать достижимые отрицательные циклы. Если из такого цикла можно пройти к цели, конечной минимальной стоимости для неё не существует. Повторение цикла позволяет уменьшать стоимость без ограничения.

Что нужно уметь после лекции

ИТОГИ

Дейкстра находит пути с минимальной суммой неотрицательных весов. Релаксация улучшает предварительную оценку и обновляет предшественника. Оценка становится окончательной в момент выбора вершины с минимальной текущей стоимостью.

При неотрицательных весах ближайшую вершину можно зафиксировать.

Проверьте понимание

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем вес одного ребра отличается от предварительного расстояния до вершины?
2. Когда меняется запись в `parent`?
3. В какой момент можно остановить поиск пути к одной цели?
4. Как неотрицательность весов используется в доказательстве?
5. Почему недостижимая вершина сохраняет оценку `inf`?

Самостоятельная работа

ПРАКТИКА

1. Увеличьте вес ребра `B → A` до 5. Заполните таблицу релаксаций и укажите два маршрута с одинаковой минимальной стоимостью.
2. Добавьте ребро `A → старт` веса 0. Выполните программу и объясните, почему цикл не снижает расстояние до старта.
3. Напишите вариант функции для одной цели. Завершайте поиск после выбора цели с минимальной оценкой и проверьте случай недостижимой цели.

Источники и документация

ЧТЕНИЕ

ОСНОВНОЕ ЧТЕНИЕ

А. Бхаргава. Грокаем алгоритмы

СПб.: Питер, 2017. Глава 7. Алгоритм Дейкстры. С. 151–181.

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