АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ЛЕКЦИЯ 9

Динамическое программирование

Состояние и переход позволяют решить каждую подзадачу один раз.

09

ИЗ 15 ЛЕКЦИЙ



МОДУЛЬ 3 Динамическое программирование

1 КУРС / 2 АКАДЕМИЧЕСКИХ ЧАСА / PYTHON 3

Цель и содержание лекции

ЦЕЛЬ И РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

определять состояния и переходы динамического программирования, вычислять оптимальную стоимость рюкзака с восстановлением набора и применять тот же метод к сравнению строк.

- Задавать состояние, базу и порядок вычислений.
- Заполнять таблицу для рюкзака 0/1.
- Извлекать ответ и оценивать стоимость таблицы.

ПЛАН ЛЕКЦИИ

1. **Подзадачи, состояния и порядок вычислений**
2. **Состояние и переход для рюкзака 0/1**
3. **Заполнение таблицы и восстановление предметов**
4. **Реализация рюкзака на Python и оценка сложности**
5. **Сравнение строк**

Две ветви перехода для рюкзака 0/1

СХЕМА



Берём максимум из допустимых вариантов. В обеих ветвях i уменьшается, предмет нельзя взять дважды.

Подзадачи, состояния и порядок вычислений

01 ТЕМА

В предыдущей лекции жадное правило для рюкзака пропустило лучший набор. Полный перебор гарантирует точный ответ, но для n предметов рассматривает 2^n подмножеств. Многие ветви перебора приводят к одному и тому же вопросу. Например, после разных решений о первых предметах может остаться одинаковая вместимость и один и тот же набор ещё не рассмотренных предметов. Результат такого расчёта можно использовать повторно.

Состояние — минимальный для выбранной модели набор параметров, который однозначно задаёт подзадачу. Её решение храним как число, логическое значение или другой результат, нужный для следующих вычислений. **Переход** описывает, как вычислить состояние по уже решённым подзадачам. **Базовые случаи** дают ответы там, где переход применять не требуется. При проектировании полезно сначала записать смысл одной ячейки словами, затем определить допустимые зависимости и только после этого строить таблицу.

Подзадачи, состояния и порядок вычислений

01 ПРОДОЛЖЕНИЕ

Табличный подход заполняет состояния в порядке от простых к сложным. Другой вариант — запустить рекурсию из нужного состояния и сохранять результат при первом вычислении. Это **мемоизация**. Оба подхода используют одинаковый переход. В табличном варианте легче увидеть порядок заполнения, а при мемоизации можно вычислять только состояния, до которых дошла рекурсия.

Состояние и переход для рюкзака 0/1

02 ТЕМА

Вместимость рюкзака `C` — неотрицательное целое число, веса предметов — положительные целые числа. Каждый предмет можно взять максимум один раз, ценности выбранных предметов складываются. Обозначим через `dp[i][c]` максимальную суммарную ценность подмножества первых `i` предметов при вместимости **не более** `c`. Рюкзак может остаться заполненным частично. В частности, для нулевой вместимости и для нулевого числа предметов ответ равен 0.

Пусть предмет `i` имеет вес `weight` и ценность `value`. Если его пропускаем, итог равен `dp[i - 1][c]`. Если он помещается и мы его берём, остаётся вместимость `c - weight` и разрешены только первые `i - 1` предметов. Стоимость этого варианта равна `value + dp[i - 1][c - weight]`. Когда оба варианта допустимы, берём максимум. Если предмет тяжелее `c`, переносим значение из предыдущей строки. Ссылка на предыдущую строку в обеих ветвях гарантирует, что текущий предмет нельзя взять дважды.

Состояние и переход для рюкзака 0/1

02 ПРОДОЛЖЕНИЕ

Эти случаи покрывают любое допустимое решение. Если лучшее решение содержит текущий предмет, остальные его предметы должны образовывать оптимальное решение для оставшейся вместимости. Иначе их можно заменить более ценным набором. Аналогично, при пропуске предмета лучший ответ должен быть оптимальным для первых `i - 1` предметов и той же вместимости. Это и обосновывает переход.

Если по условию надо занять вместимость ровно `C`, изменится смысл ячейки. При нуле доступных предметов вместимость 0 заполнить можно, а вместимость 1 и выше — невозможно. Эти состояния надо различать, например хранить минус бесконечность для невозможного случая. Для единственной гитары весом 1 и вместимости 2 ответ с ограничением «не более 2» равен 1500. При условии «ровно 2» решения нет. Этот пример показывает, как небольшая правка постановки влияет на базовые случаи и трактовку ответа.

Заполнение таблицы и восстановление предметов

Возьмём данные главы 9. Гитара весит 1 фунт и стоит 1500, магнитофон весит 4 фунта и стоит 3000, ноутбук весит 3 фунта и стоит 2000. Вместимость равна 4. Вес каждого предмета и вместимость выражены в одинаковых единицах.

Доступные предметы	0	1	2	3	4
Нет предметов	0	0	0	0	0
Добавлена гитара	0	1500	1500	1500	1500
Добавлен магнитофон	0	1500	1500	1500	3000
Добавлен ноутбук	0	1500	1500	2000	3500

Заполнение таблицы и восстановление предметов

03 ПРОДОЛЖЕНИЕ

В ячейке `dp[3][4]` при пропуске ноутбука получается 3000. При выборе ноутбука остаётся 1 фунт, и в ячейке `dp[2][1]` уже записано 1500. Поэтому второй вариант даёт `2000 + 1500 = 3500`. При восстановлении видим, что `dp[3][4]` больше `dp[2][4]`. Значит, берём ноутбук и переходим в `dp[2][1]`. Там значение совпадает с `dp[1][1]`, поэтому пропускаем магнитофон. В следующей строке разность значений показывает, что нужна гитара. При равенстве с предыдущей строкой предмет можно пропустить и сохранить оптимальную стоимость.

Реализация рюкзака на Python и оценка сложности

04 ТЕМА

Каждый элемент входного списка состоит из имени, веса и ценности. Таблица создаётся списковым включением, чтобы её строки были независимыми объектами.

Выражение `[[0] * (capacity + 1)] * (len(items) + 1)` создаёт несколько ссылок на один список. При записи в одну строку изменились бы ячейки того же столбца во всех строках. Тогда переход начал бы читать значения, уже изменённые при работе с текущим предметом. Отдельные списки позволяют явно сохранить результаты каждой подзадачи.

Реализация рюкзака на Python и оценка сложности

09_01.py / фрагмент 1 из 2

```
def knapsack(items, capacity):
    dp = [[0] * (capacity + 1) for _ in range(len(items) + 1)]

    for i, (_, weight, value) in enumerate(items, start=1):
        for limit in range(capacity + 1):
            dp[i][limit] = dp[i - 1][limit]
            if weight <= limit:
                take = value + dp[i - 1][limit - weight]
                dp[i][limit] = max(dp[i][limit], take)

    selected = []
    limit = capacity
    for i in range(len(items), 0, -1):
        if dp[i][limit] > dp[i - 1][limit]:
            name, weight, _ = items[i - 1]
            selected.append(name)
            limit -= weight

    return dp[-1][capacity], selected[::-1]
```

Реализация рюкзака на Python и оценка сложности

04 ПРИМЕР PYTHON

09_01.py / фрагмент 2 из 2

```
items = [('гитара', 1, 1500), ('магнитофон', 4, 3000),
         ('ноутбук', 3, 2000)]
print(knapsack(items, 4))
print(knapsack(items, 0))
```

РЕЗУЛЬТАТ ЗАПУСКА

```
(3500, ['гитара', 'ноутбук'])
(0, [])
```

Реализация рюкзака на Python и оценка сложности

04 ПРОДОЛЖЕНИЕ

Таблица содержит `(n + 1) × (C + 1)` ячеек, на каждую требуется постоянное число операций. При положительных `n` и `C` время и память составляют `O(nC)`. Это **псевдополиномиальная** оценка, поскольку зависит от числового значения `C`. Увеличение `C` вдвое добавляет примерно вдвое больше столбцов, хотя запись числа в двоичном виде удлиняется примерно на один разряд. Если нужна только оптимальная ценность, можно хранить один массив на `C + 1` позиций. Для каждого предмета вместимости в таком массиве обходят по убыванию. При прямом обходе ячейка с меньшей вместимостью могла бы уже включать текущий предмет.

Общая подстрока и общая подпоследовательность

05 ТЕМА

Подстрока состоит из соседних символов. Для `hish` и `fish` общая подстрока `ish` имеет длину 3, а для `hish` и `vista` общая подстрока `is` имеет длину 2. Пусть `cell[i][j]` хранит длину совпадающего окончания двух префиксов длины `i` и `j`. Если последние символы совпадают, продолжаем предыдущее совпадение и прибавляем 1 к `cell[i - 1][j - 1]`. Иначе записываем 0. Например, для `hish` и `fish` диагональ совпадений заканчивается значениями 1 для `i`, 2 для `is` и 3 для `ish`. Ответ берём как максимум по всей таблице. Совпадение может завершиться до конца одной или обеих строк.

Общая подстрока и общая подпоследовательность

05 ПРИМЕР PYTHON

09_02.py

```
def longest_common_substring(first, second):
    cell = [[0] * (len(second) + 1) for _ in range(len(first) + 1)]
    best_length = 0
    best_end = 0
    for i, left in enumerate(first, start=1):
        for j, right in enumerate(second, start=1):
            if left == right:
                cell[i][j] = cell[i - 1][j - 1] + 1
                if cell[i][j] > best_length:
                    best_length = cell[i][j]
                    best_end = i
    return first[best_end - best_length:best_end]


print(longest_common_substring('hish', 'fish'))
print(longest_common_substring('hish', 'vista'))
```

РЕЗУЛЬТАТ ЗАПУСКА

```
ish
is
```

Общая подстрока и общая подпоследовательность

05 ПРОДОЛЖЕНИЕ

В подпоследовательности символы могут идти с пропусками, сохраняя исходный порядок. Например, `fsh` — общая подпоследовательность `fosh` и `fish`. Здесь состояние хранит длину лучшей подпоследовательности **внутри** двух префиксов. Если последние символы совпадают, к диагональному значению прибавляется 1. При несовпадении берётся максимум ячеек сверху и слева, что соответствует пропуску символа одной из строк. Ответ находится в правом нижнем углу. Обе таблицы для строк длин `m` и `n` заполняются за `O(mn)` для непустых строк.

Что нужно уметь после лекции

ИТОГИ

Смысл состояния определяет переход и место, откуда нужно брать ответ. В рюкзаке каждая строка добавляет возможность выбрать ещё один предмет. При сравнении строк выбор между подстрокой и подпоследовательностью меняет правило обработки несовпадающих символов.

> Состояние и переход позволяют решить каждую подзадачу один раз.

Проверьте понимание

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что означает ячейка `dp[i][c]` и какие решения она охватывает?
2. Почему при добавлении предмета нужно обращаться к предыдущей строке?
3. Как восстановить один из оптимальных наборов при равных ценностях вариантов?
4. Почему время `O(nC)` называют псевдополиномиальным?
5. Как различаются переходы при несовпадении символов для подстроки и подпоследовательности?

Самостоятельная работа

ПРАКТИКА

1. Добавьте к рюкзаку телефон весом 1 и ценностью 2000. Заполните новую строку таблицы и восстановите набор для вместимости 4.
2. Реализуйте вариант рюкзака с одним массивом. Сравните его с табличным алгоритмом на исходном примере и на рюкзаке, в который один предмет мог бы поместиться дважды по весу.
3. Вручную сравните `fosh` с `fish` и `fort` по длине общей подстроки и общей подпоследовательности. Объясните, почему эти критерии дают разные выводы.

Источники и документация

ЧТЕНИЕ

ОСНОВНОЕ ЧТЕНИЕ

А. Бхаргава. Грокаем алгоритмы

СПб.: Питер, 2017. Глава 9. Динамическое программирование. С. 206–235.

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