

Динамическое программирование

Динамическое программирование — это когда у нас есть задача, которую непонятно как решать, и мы разбиваем ее на меньшие задачи, которые тоже непонятно как решать. (с) А. Кумок.

Задача про черепашку

Есть клетчатое поле $N \times M$. В левом верхнем углу сидит черепашка. Она умеет ходить только вправо или вниз.

А) Сколько у неё разных путей до правого нижнего угла?

Первая основная идея ДП

Будем искать ответ не только на нашу общую задачу, но и на более мелкие аналогичные задачи («подзадача»). В нашем случае решим для каждой клетки поля сколькими способами до неё можно добраться.

А что дальше?

- 1) Ответ для верхнего левого угла очевиден. У нас только один способ до него добраться.
- 2) Для клеток левого столбца и верхней строки тоже всё очевидно.

Вторая основная идея ДП

Решая задачу для очередной клетки, будем считать, что мы уже знаем ответ для предыдущих клеток и попробуем, используя это знание, найти ответ для текущей.

А как мы можем попасть в
очередную клетку?

Всё очевидно! Мы можем прийти в неё
либо с верхней, либо с левой клетки.

$$\text{Answer}[i][j] = \text{Answer}[i - 1][j] + \text{Answer}[i][j - 1]$$

Данную задачу можно решить также с помощью комбинаторики:

$$\binom{i-1}{i+j-2} = \binom{j-1}{i+j-2} = \text{Ans}[i][j]$$

Б) Пусть в каждой клетке поля записано некоторое число. Требуется найти максимальную сумму чисел, которую можно набрать по пути в правый нижний угол.

34	17	27	43	53
31	30	42	49	11
18	31	21	24	55
43	41	36	58	53

Пусть $A[i][j]$ – число в (i,j) клетке, а $dp[i][j]$ ответ для неё.

$$1) dp[1][1] = A[1][1];$$

$$2) dp[1][j] = \sum_{k=1}^j A[1][k],$$

$$dp[i][1] = \sum_{k=1}^i A[k][1]$$

$$3) dp[i][j] = \max(dp[i-1][j], dp[i][j-1]) + A[i][j];$$

В результате получим такую таблицу:

34	51	78	121	174
65	95	137	186	197
83	126	158	210	265
126	167	203	268	321

Последовательность из нулей и единиц без двух единиц подряд.

Дано число N . Рассмотрим все 2^N последовательностей из нулей и единиц. Назовём последовательность хорошей, если в ней нигде не встречается две единицы подряд. Требуется посчитать общее количество хороших последовательностей.

- 1) $dp[i][j]$ – ответ для последовательности длины i , оканчивающейся на j
- 2) $dp[1][0] = dp[1][1] = 1;$
- 3) $dp[i][0] = dp[i - 1][0] + dp[i - 1][1];$
- 4) $dp[i][1] = dp[i - 1][1];$

А можно заметить, что это числа Фибоначчи 😊

Немного теории

Определения:

- 1) Состояние – это набор параметров, с помощью которых мы фиксируем подзадачу
- 2) Переходы – это рекуррентное соотношение между состояниями
- 3) Порядок пересчёта – это порядок, по которому мы обходим состояния

Чтобы успешно решить задачу динамикой нужно ответить на следующие вопросы:

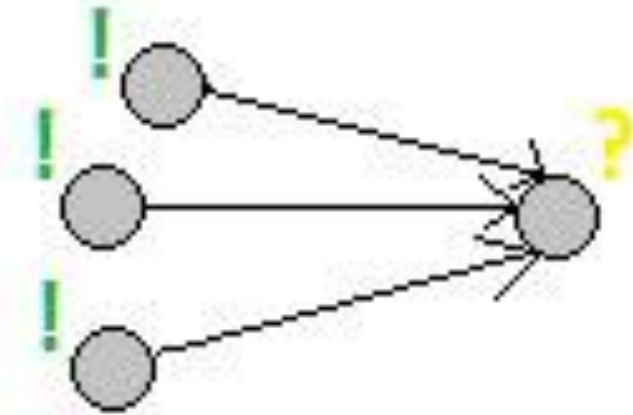
- 1) Что мы вычисляем?
- 2) Какие у нас состояния?
- 3) Каковы значения в начальных состояниях?
- 4) Какие переходы между состояниями?
- 5) Каков порядок пересчёта?
- 6) Где хранится ответ на задачу?

Порядок пересчёта

Существует три порядка пересчёта:

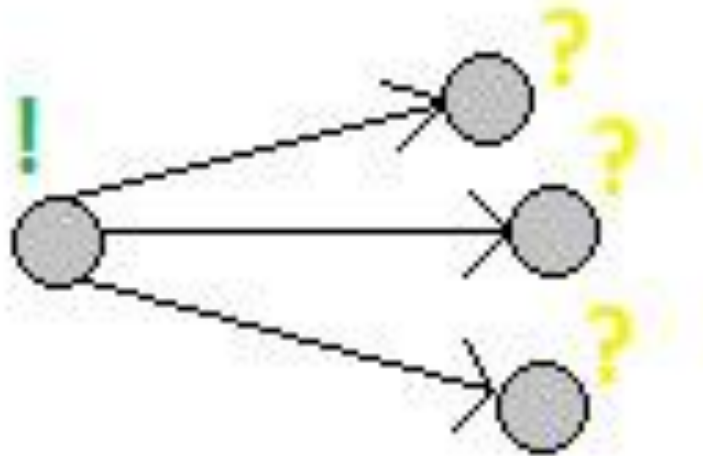
1) Прямой

Состояния последовательно пересчитываются исходя из уже посчитанных



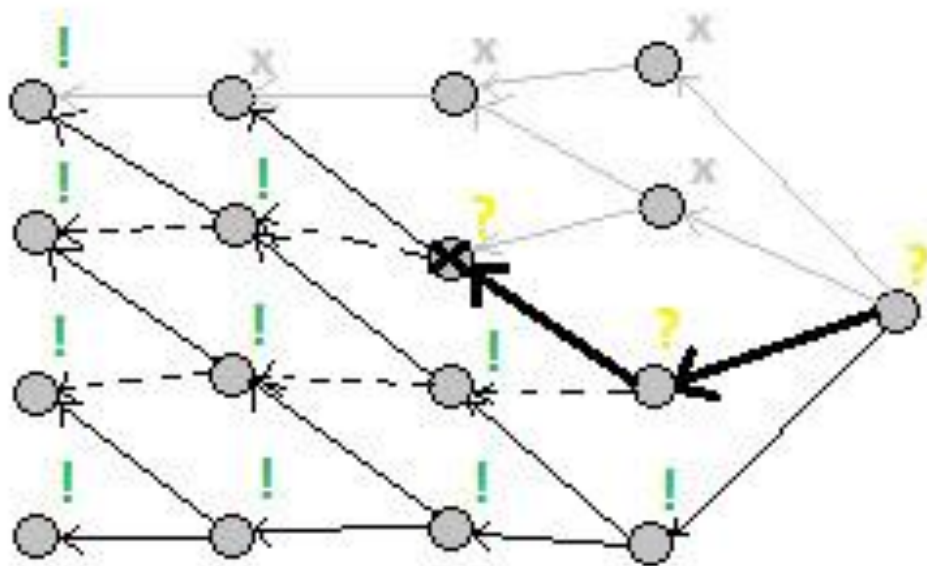
2) Обратный

Обновляются все состояния, зависящие от текущего состояния



3) Ленивая динамика

Рекурсивная функция пересчёта динамики. Это что-то вроде поиска в глубину по ациклическому графу состояний, где рёбра – это зависимости между ними.



Классы задач на ДП

- 1) Подсчёт объектов, в том числе определение существования объекта.**
Т.е. надо посчитать, сколько всего существует объектов с заданными свойствами, или проверить, существует ли хотя бы один.
- 2) Нахождение оптимального объекта.**
Требуется в некотором множестве объектов найти в некотором смысле оптимальный.
- 3) Вывод k-ого объекта.** Нужно найти в некотором порядке k-ый объект.

Виды задач на ДП

- 1) Линейное ДП
- 2) Многомерное ДП
- 3) ДП на подотрезках
- 4) ДП по полной сумме
- 5) ДП на ациклических графах
- 6) ДП на деревьях
- 7) Игры(ретро-анализ)
- 8) ДП на подмножествах.
- 9) ДП по профилю
- 10) Динамика по изломанному профилю

Отдельно рассмотрим семейство задач о рюкзаке

Классическая постановка задачи:

Есть N предметов, обладающих весом w_i и стоимостью p_i . В рюкзак влезают предметы, суммарный вес которых не превосходит W . Какую максимальную ценность может иметь рюкзак?

- Перебирать все подмножества набора из N предметов. Сложность такого решения $O(2^N)$.
- Метод динамического программирования. Сложность – $O(N \times W)$.

Решение методом динамического программирования

Пусть $dp[i][j]$ – есть максимальная стоимость предметов, которое можно уложить в рюкзак вместимости j , если мы используем только первые i предметов.

$$dp[i][0] = dp[0][j] = 0;$$

$$dp[i][j] = \max(dp[i-1][j], dp[i-1][j - w_i] + p_i);$$

Ответ: $dp[N][W]$;

Пример

$$W = 13, N = 5$$

$$w_1 = 3, p_1 = 1$$

$$w_2 = 4, p_2 = 6$$

$$w_3 = 5, p_3 = 4$$

$$w_4 = 8, p_4 = 7$$

$$w_5 = 9, p_5 = 6$$

	1	2	3	4	5	6	7	8	9	10	11	12	13
k=0	0	0	0	0	0	0	0	0	0	0	0	0	0
k=1	0	0	1	1	1	1	1	1	1	1	1	1	1
k=2	0	0	1	6	6	6	7	7	7	7	7	7	7
k=3	0	0	1	6	6	6	7	7	10	10	10	11	11
k=4	0	0	1	6	6	6	7	7	10	10	10	13	13
k=5	0	0	1	6	6	6	7	7	10	10	10	13	13

Другие задачи семейства

Ограниченный рюкзак - обобщение классической задачи, когда любой предмет может быть взят некоторое количество раз.

Пример: Вор грабит склад. Он может унести ограниченный вес, каждый товар на складе содержится в определенном ограниченном количестве. Нужно унести предметов на максимальную сумму.

Варианты решения:

- Перебор.
- Методом динамического программирования.

Решение методом ДП

Пусть $dp[i][j]$ – максимальная стоимость любого возможного числа предметов типов от 1 до i , суммарным весом до j .

$$dp[0][j] = dp[i][0] = 0;$$

$$dp[i][j] = \max(dp[i-1][j], dp[i-1][j-l*w_i]+l*p_i), \text{ где } l \in [0, \min(b_i, \lfloor c/w_i \rfloor)]$$

Ответ: $dp[N][W]$

Сложность: $O(NW^2)$

Неограниченный рюкзак - обобщение ограниченного рюкзака, в котором любой предмет может быть выбран любое количество раз.

Пример: Перекупщик закупается на оптовой базе. Он может увезти ограниченное количество товара, количество товара каждого типа на базе не ограничено. Нужно увезти товар на максимальную сумму.

Варианты решения:

- Перебор.
- Методом динамического программирования.

Пусть $dp[i][j]$ – есть максимальная стоимость предметов, которое можно уложить в рюкзак вместимости j , если мы используем только первые i предметов.

$$dp[i][0] = dp[0][j] = 0;$$

$$dp[i][j] = \max(dp[i-1][j], dp[i][j - w_i] + p_i);$$

Ответ: $dp[N][W]$;

Непрерывный рюкзак - вариант задачи, в котором возможно брать любую дробную часть от предмета, при этом удельная стоимость сохраняется.

Пример: Вор грабит мясника. Суммарно он может унести ограниченный вес товаров. Вор может резать товары без ущерба к удельной стоимости. Нужно унести товара на максимальную сумму.

Варианты решения:

Возможность брать любую часть от предмета сильно упрощает задачу. Жадный алгоритм дает оптимальное решение в данном случае.

Мультипликативный рюкзак – есть N предметов и M рюкзаков ($M \leq N$). У каждого рюкзака своя вместимость W_i . Задача: выбрать M не пересекающихся множеств, назначить соответствие рюкзакам так, чтобы суммарная стоимость была максимальна, а вес предметов в каждом рюкзаке не превышал его вместимость.

Пример: У транспортной компании есть парк машин разной грузоподъемности. Нужно перевезти товара на максимальную сумму с одного склада на другой одновременно.

Варианты решения:

Перебор

Полезные ссылки

goo.gl/1p0tXp – Подробная лекция о динамическом программировании

goo.gl/ehm0lw – Задача о рюкзаке