Динамическое программирование

Руслан Назирович Мокаев

Математико-механический факультет,
Санкт-Петербургский государственный университет

Санкт-Петербург, 14.05.2024

Содержание лекции

- Динамическое программирование, метод разбиения и жадные алгоритмы.
- Отличия семейств алгоритмов.
- Этапы алгоритма динамического программирования. Принцип оптимальности Беллмана и перекрывающиеся вспомогательные задачи.
- Задача о минимальном размене и задача и количестве разменов.

Задача оптимизации и методы решения

$$f(x) \to \min,$$

при некоторых ограничениях (зависят от задачи).

Динамическое программирование (ДП) – методика решения поставленной задачи путем комбинирования решений вспомогательных задач.

Метод разбиения (декомпозиции, "разделяй и влавствуй"):

- задача разбивается на более мелкие вспомогательные задачи;
- эти вспомогательные задачи рекурсивно решаются;
- их решения комбинируются и образуют решение исходной задачи.

Большое количество сортировок, алгоритмы перемножения больших чисел, важнейшего алгоритма быстрого преобразования Фурье, вычисления дискретного преобразования Фурье.

Жадные алгоритмы: на каждом очередном шаге совершается выбор, который кажется самым оптимальным в данный момент.

Алгоритмы Краскала, Прима-Ярника и разработка кода Хаффмана.

Этапы алгоритма ДП

1. Описание структуры оптимального решения;
2. Рекурсивное определение значения, соответствующего оптимальному решению;
3. Вычисление значения, соответствующего оптимальному решению, с помощью метода восходящего анализа;
4. Составление оптимального решения на основе информации, полученной на предыдущих этапах (составлять оптимальное решение нужно не во всех задачах; если все же требуется, то на шаге 3 вводится дополнительная структура для хранения информации).

Отличия парадигм ДП и "разделяй и влавствуй"

- Алгоритм ДП по-разному определяет способы разделения входной задачи на мелкие подзадачи; рекурсивные вызовы в методе разбиения придерживаются одного способа разделения входа на вспомогательные задачи;
- Поскольку способы разделения различны, то вспомогательные задачи зачастую повторяются (мы увидим это в примерах). Поэтому для оптимизации их решения кэшируются в специальных структурах (очень часто это таблицы). В методе разбиения вспомогательные задачи не пересекаются и кэширование их решения не нужно;
- Обычно понижение в сложности, которое получается заменой условного "простого" алгоритма, заметнее в случае ДП;
- Обычно в методе разбиения размеры вспомогательных задач кратно меньше размеры входной задачи, в то время как в алгоритмах ДП размера могут едва отличаться.

Принцип оптимальности Беллмана

Ключевые свойства, которые должны быть у задачи оптимизации:

- Наличие оптимальной подструктуры;
- Перекрывающиеся вспомогательные программы.

Принцип оптимальности Беллмана, или наличие оптимальной структуры, заключается в том, что в оптимальном решении задачи оптимизации содержатся оптимальные решения вспомогательных подзадач.

Наличие оптимальной подструктуры роднит ДП с жадным подходом, в котором ее наличие тоже необходимо. Отличия возникают в *направлении использования* оптимальной подструктуры:

- в ДП используется в восходящем направлении: из решения вспомогательных задач конструируется решение исходной – движение "снизу вверх";
- в жадных алгоритмах движение по нисходящей: вместо выбора из нескольких решений выбирается кажущийся оптимальным на данный момент, а потом уже решается возникшая в результате выбора вспомогательная задача – движение "сверху вниз".

Перекрывающиеся вспомогательные задачи: пространство вспомогательных задач должно быть *небольшим* (обычно это полиномиальная функция от размера входных данных) в том смысле, что в результате выполнения рекурсивного алгоритма одни и те же вспомогательные задачи решаются снова и снова (возникают в процессе решения задачи в разных подзадачах и в этом смысле **перекрываются**), а новые вспомогательные задачи не возникают.

- такое количество все же ведет к корректному решению задачи и позволяет ускорить решение в сравнение с условным "простым" решением;
- каждая вспомогательная задача решается ровно один раз, после чего ее решение записывается в специальную таблицу. Из этой таблицы данные уже потом извлекаются за константное время.

Определение: Процесс сохранения результатов вызова функции на определенных параметрах для предотвращения повторных вызовов называется **мемоизацией** (англ. memoization) или **запоминанием**.

Примеры перекрываемости:

- сортировка слиянием;
- вычисление чисел Фибоначчи.

Пример: задача о минимальном размене

Задача о минимальном размене: имеется некоторая сумма $N \in \mathbb{N}$, которую необходимо разменять наименьшим количеством монет номиналами $\{w_i\}_{i=1}^{n}$, $w_i \in \mathbb{N}$. Эту задачу можно записать как задачу линейного программирования (оптимизации):

$$f(N) = \sum_{i=1}^{n} x_i \to \min_{\Omega}$$

$$\Omega \begin{cases} \sum_{i=1}^{n} x_i w_i = N \\ x_i \geq 0, \ i \in 1:n, \end{cases}$$

где x_i – количество монет номинала w_i.

Эта задача, в свою очередь, является подзадачей **задачи о количестве разменов** (англ. coin-making problem): дано значение N и набор номиналов монет $\{w_i\}_{i=1}^{n}$. Необходимо посчитать (и при необходимости вывести) количество способов разменять значение N монетами $\{w_i\}_{i=1}^{n}$.

Задача о наименьшем размене демонстрирует оптимальную подструктуру в следующем виде: рассмотрим оптимальный размен $\{x_i\}_{i=1}^k$ монетами $\{w_i\}_{i=1}^k$ суммы N. Возьмем его разбиение мощности 2:

$$b = \sum_{i=1}^{k} \bar{x}_i w_i, \;\; N - b = \sum_{i=1}^{k} (x_i - \bar{x}_i) w_i,$$

где $0 \leq \bar{x}_i \leq x_i, i \in \overline{1:k}$. Тогда $\{\bar{x}_i\}_{i=1}^k$ – это оптимальный размен b монетами $\{w_i\}_{i=1}^k$, а $\{x_i - \bar{x}_i\}_{i=1}^k$ – оптимальный размен $N - b$ монетами $\{w_i\}_{i=1}^k$.

Док-во: от противного. Пусть существует $\{x_i^*\}_{i=1}^k$ – более оптимальный способ разменять b: $b = \sum_{i=1}^k x_i^* w_i, \;\; \sum_{i=1}^k x_i^* < \sum_{i=1}^k \bar{x}_i$. Тогда

$$\sum_{i=1}^{k} x_i^* w_i + \sum_{i=1}^{k} (x_i - \bar{x}_i) w_i = b + (N - b) = N,$$

$$\sum_{i=1}^{k} x_i^* + \sum_{i=1}^{k} (x_i - \bar{x}_i) < \sum_{i=1}^{k} \bar{x}_i + \sum_{i=1}^{k} (x_i - \bar{x}_i) = \sum_{i=1}^{k} x_i \quad (?!)$$

Рекурсивное определение оптимального решения

Обозначим за $c[p]$ минимальное количество монет номиналами $\{w_i\}_{i=1}^{n}$, необходимых для размена суммы p. Заметим, что в оптимальном разбиении суммы p должен присутствовать номинал w_i такой, что $w_i < p$. Следовательно, оставшиеся монеты в оптимальном размене должны образовывать оптимальный размен суммы $p - w_i$ и тогда справедливо соотношение

$$c[p] = 1 + c[p - w_i].$$

В общем случае мы не знаем какой именно номинал входит в оптимальное решение, однако мы можем проверить все монеты, удовлетворяющие свойству $w_i < p$. Таких монет может быть несколько, поэтому проверяем все подходящие номиналы и в качестве $c[p]$ берем минимальное из всех $1 + c[p - w_i]$. Отметим также, что если необходимо разменять нулевую сумму, то для этой цели нужно ровно ноль монет. Теперь составим рекурсивное соотношение:

$$c[p] = \begin{cases} 0, \text{ если } p = 0, \\ \min\limits_{i:w_i<p} \big(1 + c[p - w_i]\big), \text{ если } p > 0. \end{cases}$$

Вычисление методом восходящего анализа

1. Проинициализируем нулями два списка $c[0:N]$ и $s[0:N]$.
   - В $c[l]$ будет храниться оптимальные решения для размена l.
   - В $s[l]$ будет храниться индекс первой монеты оптимального размена l.
2. Пробегаемся по всем потенциальным промежуточным суммам l и
   - присваиваем значению $c[l] = \min\limits_{i:w_i<l} \big(1 + c[l - w_i]\big)$ для всех $w_i < l$ согласно рекурсивному соотношению.
   - присваиваем значению $s[l]$ индекс номинала i, на котором был достигнут $\min\limits_{i:w_i<l} \big(1 + c[l - w_i]\big)$.
3. После заполнения списков c и s значение $c[N]$ будет равняться искомому оптимуму.

Заметим, что по построению справедливо, что в $c[l]$, $0 \leq l \leq N$ будет храниться оптимальные решения для размена l, а в $s[l]$, $0 \leq l \leq N$ будет храниться индекс первой монеты оптимального размена l.

Составление оптимального решения

Для восстановления решения необходимо просмотреть массив $s[N]$ следующим образом:

- по построению $s[N]$ содержит индекс первой монеты оптимального размена суммы N; далее переходим к рассмотрению суммы $N - w_{s[N]}$;
- на промежуточном шаге рассматриваем размен суммы l: первая монета ее оптимального размена это $w_{s[l]}$ и далее переходим к рассмотрению суммы $l - w_{s[l]}$;
- выполняем этот шаг пока не спустимся до нуля.

Задача о количестве разменов

Введем $h[k, v]$ – количество разменов суммы v монетами с номиналами $\{w_i\}_{i=1}^{k}$ (они отсортированы в порядке возрастания номиналов). Рекурсивное соотношение основывается на том, что размены суммы v можно поделить на два множества: содержащие наибольший номинал w_k и не содержащие (в них наибольший номинал не превышает w_{k-1}).

Тогда количество разменов в первом множестве это ровно $h[k-1, v]$, а во втором будет $h[k, v - w_k]$:

$$h[k, v] = \begin{cases} h[k-1, v] + h[k, v - w_k], & \text{если } v \geq w_k, \\ h[k-1, v], & \text{если } v < w_k. \end{cases}$$

При этом считаем (граничные условия), что

- если сумма, которую необходимо разменять, равна 0, то есть ровно один вариант ее размена ($x_i = 0$);
- если сумма $v \neq 0$, то $h[0, v] = 0$ (ноль способов разменять сумму).

Значение $h[n, N]$ будет равно количеству разменов суммы N всеми доступными номиналами!

	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	1	0	1	0	1	0	1
3	1	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	1	0	1	0	1	0	1
3	1	0	1	1	1	1	2	1	2	2	2
5	1	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	1	0	1	0	1	0	1
3	1	0	1	1	1	1	2	1	2	2	2
5	1	0	1	1	1	2	2	2	3	3	4
6	1	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	1	0	1	0	1	0	1
3	1	0	1	1	1	1	2	1	2	2	2
5	1	0	1	1	1	2	2	2	3	3	4
6	1	0	1	1	1	2	3	2	4	4	5

Второе решение задачи о минимальном размене

Обозначим за $S_{k,v}$ минимальный размен суммы v на монеты с номиналами $\{w_i\}_{i=1}^{k}$. Как и в задаче о количестве разменов возможны два случая:

- если $w_k \notin S_{k,v}$, то $|S_{k,v}| = |S_{k-1,v}|$; ведь если $|S_{k,v}| < |S_{k-1,v}|$, то минимальный размен $S_{k-1,v}$ подходит для размена v и по мощности он меньше $S_{k,v}$ – противоречие с оптимальностью $S_{k,v}$. А если $|S_{k,v}| > |S_{k-1,v}|$, то минимальный размен $S_{k,v}$ подходит для размена v на монеты $\{w_i\}_{i=1}^{k-1}$ (т.к. в нем нет w_k) – противоречие с оптимальностью $S_{k-1,v}$.
- если $w_k \in S_{k,v}$, то оптимальный размен $S_{k,v-w_k}$ будет содержать ровно на одну монету меньше, нежели $S_{k,v}$ (это монета номиналом w_k). Если это не так, то, по аналогии с предыдущим пунктом, получим противоречие с оптимальностью либо $S_{k,v}$, либо $S_{k,v-w_k}$.

Нельзя наперед знать будет ли $w_k \in S_{k,v} \Rightarrow$ для нахождения $S_{k,v}$ нужно взять наименьшее по мощности множество из $S_{k-1,v}$ и $S_{k,v-w_k} \cup \{w_k\} \Rightarrow$ задача демонстрирует наличие оптимальной подструктуры!

Теперь рекурсивно определим оптимальное решение. По аналогии с задачей о количестве разменов введем обозначение $c[k, v] := |S_{k,v}|$ – мощность оптимального размена суммы v монетами с номиналами $\{w_i\}_{i=1}^{k}$.

Заметим, что если $v < w_k$, то монета с номиналом w_k точно не участвует в оптимальном размене v на $\{w_i\}_{i=1}^{k}$ и $c[k, v] = |S_{k,v}| = |S_{k-1,v}| = c[k-1, v]$.

Если $v < w_k$, то, по рассуждению о существовании оптимальной подструктуры, придется выбирать минимум из $|S_{k-1,v}| = c[k-1, v]$ и $|S_{k,v-w_k} \cup \{w_k\}| = c[k, v - w_k] + 1$.

Таким образом, алгоритм базируется на рекурсивном соотношении

$$c[k, v] = \begin{cases} c[k-1, v], \text{ если } v < w_k, \\ \min\big(c[k-1, v], c[k, v - w_k] + 1\big), \text{ если } v \geq w_k, \end{cases}$$
$$c[0, v] = +\infty, v > 0,$$
$$c[k, 0] = 0.$$

Вычисление методом восходящего анализа

- Инициализируем таблицу $c[m, W]$ нулями, в самом верхнем ряду на каждой позиции (кроме нулевой) стоят $+\infty$. Строки будут соответствовать номиналам, отсортированным по возрастанию; столбцы – размениваемым суммам от 0 до W.
- Далее на каждом шаге добавляем в рассмотрение одну монету и проводим пересчет минимальных количеств монет, необходимых для размена значений во всем ряду с учетом вновь добавленного номинала. Т.е., на i-ой итерации мы будем заполнять i-ю строку.
- Принцип заполнения клетки $c[i, v]$ следующий:
  - Если $v < w[i] \Rightarrow$ в клетку записывается ответ из клетки выше
    $$c[i, v] = c[i - 1, v],$$
    после чего переходим в соседнюю клетку;
  - Если $v \geq w[i] \Rightarrow$ возможны размены без использования i-ой монеты и с использованием $\Rightarrow$ в таблицу запишем:
    $$c[i, v] = \min\big(c[i - 1, v], c[i, v - w[i]] + 1\big).$$
- После заполнения очередной строки переходим к заполнению следующей.
- После заполнения всей таблицы ответ будет находиться в $c[n, W]$.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	1	∞	2	∞	3	∞	4	∞	5	∞	6	∞	7	∞	8	∞	9	∞	10	∞	11	∞	12	∞	13	∞	14	∞
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	1	∞	2	∞	3	∞	4	∞	5	∞	6	∞	7	∞	8	∞	9	∞	10	∞	11	∞	12	∞	13	∞	14	∞
5	0	∞	1	∞	2	1	3	2	4	3	2	4	3	5	4	3	5	4	6	5	4	6	5	7	6	5	7	6	8	7
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	1	∞	2	∞	3	∞	4	∞	5	∞	6	∞	7	∞	8	∞	9	∞	10	∞	11	∞	12	∞	13	∞	14	∞
5	0	∞	1	∞	2	1	3	2	4	3	2	4	3	5	4	3	5	4	6	5	4	6	5	7	6	5	7	6	8	7
10	0	∞	1	∞	2	1	3	2	4	3	1	4	2	5	3	2	4	3	5	4	2	5	3	6	4	3	5	4	6	5
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	∞	1	∞	2	∞	3	∞	4	∞	5	∞	6	∞	7	∞	8	∞	9	∞	10	∞	11	∞	12	∞	13	∞	14	∞
5	0	∞	1	∞	2	1	3	2	4	3	2	4	3	5	4	3	5	4	6	5	4	6	5	7	6	5	7	6	8	7
10	0	∞	1	∞	2	1	3	2	4	3	1	4	2	5	3	2	4	3	5	4	2	5	3	6	4	3	5	4	6	5
20	0	∞	1	∞	2	1	3	2	4	3	1	4	2	5	3	2	4	3	5	4	1	5	2	6	3	2	4	3	5	4