Динамическое программирование: задача о рюкзаке

Александр Куликов

Онлайн-курс «Алгоритмы: теория и практика. Методы»
http://stepic.org/217

Задача о рюкзаке

Вход: веса $w_1, \ldots, w_n \in \mathbb{N}$ и стоимости $c_1, \ldots, c_n \in \mathbb{N}$ данных n предметов; вместимость рюкзака $W \in \mathbb{N}$.

Выход: максимальная стоимость предметов суммарного веса не более W.

Варианты

- Рюкзак с повторениями: неограниченное количество каждого из предметов.
- Рюкзак без повторений: единственный экземпляр каждого предмета.

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

Пример: $W = 10$

30 руб. 14 руб. 16 руб. 9 руб.

6	3	4	2

30 9 9

6	2	2

всего: 48 руб.

с повторениями

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

30	9	9	
6	2	2	всего: 48 руб.

с повторениями

30	16	
6	4	всего: 46 руб.

без повторений

Рюкзак с повторениями: подзадачи

- Рассмотрим оптимальное решение и предмет i в нём:

Рюкзак с повторениями: подзадачи

- Рассмотрим оптимальное решение и предмет i в нём:

- Если вытащить данный предмет из рюкзака, то мы получим оптимальное заполнение рюкзака вместимости $W - w_i$ («вырезать и вставить»).

Рюкзак с повторениями: подзадачи

- Рассмотрим оптимальное решение и предмет i в нём:

- Если вытащить данный предмет из рюкзака, то мы получим оптимальное заполнение рюкзака вместимости $W - w_i$ («вырезать и вставить»).
- Подзадачи:

$$D[w] = \text{макс. стоимость рюкзака вместимости } w.$$

Рюкзак с повторениями: подзадачи

- Рассмотрим оптимальное решение и предмет i в нём:

- Если вытащить данный предмет из рюкзака, то мы получим оптимальное заполнение рюкзака вместимости $W - w_i$ («вырезать и вставить»).
- Подзадачи:

$$D[w] = \text{макс. стоимость рюкзака вместимости } w\,.$$

- Тогда

$$D[w] = \max_{i\colon w_i \le w} \left\{ D[w - w_i] + c_i \right\}.$$

Дин. прог. снизу вверх

Функция
KnapsackWithRepsBU($W, w_1, \ldots, w_n, c_1, \ldots, c_n$)

создать массив $D[0 \ldots W] = [0, 0, \ldots, 0]$
для w от 1 до W:
  для i от 1 до n:
    если $w_i \leq w$:
      $D[w] \leftarrow \max(D[w], D[w - w_i] + c_i)$
вернуть $D[W]$

Дин. прог. снизу вверх

Функция $\textsc{KnapsackWithRepsBU}(W, w_1, \ldots, w_n, c_1, \ldots, c_n)$

создать массив $D[0 \ldots W] = [0, 0, \ldots, 0]$
для w от 1 до W:
  для i от 1 до n:
    если $w_i \leq w$:
      $D[w] \leftarrow \max(D[w], D[w - w_i] + c_i)$
вернуть $D[W]$

Время работы: $O(nW)$.

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

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

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

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

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

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

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

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

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	0	0	0	0	0	0	0

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	0	0	0	0	0	0	0

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	18	0	0	0	0	0	0

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	18	23	30	32	39	44	48

Рюкзак без повторений

- Что если повторения запрещены?
- Знание оптимальных стоимостей для $D[w - w_i]$ не поможет для вычисления $D[w]$, поскольку оптимальное решение для рюкзака вместимости $w - w_i$ уже может содержать i-й предмет (и тогда к этому решению нельзя будет просто добавить предмет i, чтобы получить решение для рюкзака вместимости w).
- Новые подзадачи: для $0 \leq w \leq W$ и $0 \leq i \leq n$, $D[w, i]$ — максимальная стоимость рюкзака вместимости w, если разрешено использовать только предметы $1, \ldots, i$.
- Предмет i либо используется, либо нет:

$$D[w, i] = \max\{D[w - w_i, i - 1] + c_i, D[w, i - 1]\}\,.$$

KnapsackWithoutRepsBU$(W, w_1, \ldots, w_n, c_1, \ldots, c_n)$

```
создать массив D[0...W, 0...n]
для w от 0 до W:
  D[w, 0] ← 0
для i от 0 до n:
  D[0, i] ← 0
для i от 1 до n:
  для w от 1 до W:
    D[w, i] ← D[w, i − 1]
    если w_i ≤ w:
      D[w, i] = max(D[w, i], D[w − w_i, i − 1] + c_i)
вернуть D[W, n]
```

$\textsc{KnapsackWithoutRepsBU}(W, w_1, \ldots, w_n, c_1, \ldots, c_n)$

```
создать массив D[0...W, 0...n]
для w от 0 до W:
  D[w, 0] ← 0
для i от 0 до n:
  D[0, i] ← 0
для i от 1 до n:
  для w от 1 до W:
    D[w, i] ← D[w, i − 1]
    если w_i ≤ w:
      D[w, i] = max(D[w, i], D[w − w_i, i − 1] + c_i)
вернуть D[W, n]
```

Время работы: $O(nW)$.

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Оптимальное решение:

1	2	3	4

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Оптимальное решение:

1	2	3	4
			0

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Оптимальное решение:

1	2	3	4
		1	0

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Оптимальное решение:

1	2	3	4
	0	1	0

Пример: $W = 10$

30 руб.	14 руб.	16 руб.	9 руб.
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Оптимальное решение:

1	2	3	4
1	0	1	0

Сверху вниз или снизу вверх?

- Рассмотренные алгоритмы заполняют таблицу снизу вверх: от более простых задач к более сложным.

Сверху вниз или снизу вверх?

- Рассмотренные алгоритмы заполняют таблицу снизу вверх: от более простых задач к более сложным.
- Алгоритм, заполняющий таблицу сверху вниз, делает рекурсивные вызовы для подзадач, но до того, как решать подзадачу, проверят, не сохранён ли уже ответ для неё в таблице.

Сверху вниз или снизу вверх?

- Рассмотренные алгоритмы заполняют таблицу снизу вверх: от более простых задач к более сложным.
- Алгоритм, заполняющий таблицу сверху вниз, делает рекурсивные вызовы для подзадач, но до того, как решать подзадачу, проверят, не сохранён ли уже ответ для неё в таблице.
- Если все подзадачи должны быть решены, то подход снизу вверх обычно работает быстрее, поскольку не имеет накладных расходов на рекурсию.

Сверху вниз или снизу вверх?

- Рассмотренные алгоритмы заполняют таблицу снизу вверх: от более простых задач к более сложным.
- Алгоритм, заполняющий таблицу сверху вниз, делает рекурсивные вызовы для подзадач, но до того, как решать подзадачу, проверят, не сохранён ли уже ответ для неё в таблице.
- Если все подзадачи должны быть решены, то подход снизу вверх обычно работает быстрее, поскольку не имеет накладных расходов на рекурсию.
- Есть, однако, ситуации, когда не нужно решать все подзадачи (чтобы решить исходную задачу): например, если W и все w_i делятся на 100, то нас не интересуют решения для подзадач $D[w]$ при w, не делящемся на 100.

Дин. прог. сверху вниз для рюкзака с повторениями

$\textsc{KnapsackTD}(w)$

```
если w нет в хеш-таблице H:
   v ← 0
   для всех i от 1 до n:
      если w_i ≤ w:
         v ← max{v, KnapsackTD(w − w_i) + c_i}
   H[w] ← v
вернуть v
```

Время работы

- Время работы $O(nW)$ не является полиномиальным, потому что длина входа пропорциональная $\log W$, а не W.

Время работы

- Время работы $O(nW)$ не является полиномиальным, потому что длина входа пропорциональная $\log W$, а не W.
- Другими словами, время работы есть $O(n2^{\log W})$.

Время работы

- Время работы $O(nW)$ не является полиномиальным, потому что длина входа пропорциональная $\log W$, а не W.
- Другими словами, время работы есть $O(n2^{\log W})$.
- Например, для

$$W = 71\,345\,970\,345\,617\,824\,751$$

(всего двадцать цифр!) алгоритму потребуется около 10^{20} базовых операций.