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.

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

Пример: W = 1030 руб. 14 руб. 16 руб. 9 руб. 30 всего: 48 руб. с повторениями

Пример: W = 1030 руб. 14 руб. 16 руб. 9 руб. 6 30 всего: 48 руб. с повторениями 30 16 всего: 46 руб. без повторений

■ Рассмотрим оптимальное решение и предмет *i* в нём:

 c_i

 W_i

■ Рассмотрим оптимальное решение и предмет *і* в нём:

C_i
W_i

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

■ Рассмотрим оптимальное решение и предмет *і* в нём:

C_i W_i

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

D[w] = макс. стоимость рюкзака вместимости w.

■ Рассмотрим оптимальное решение и предмет *і* в нём:

C_i W_i

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

D[w] = макс. стоимость рюкзака вместимости w.

Тогда

$$D[w] = \max_{i: w_i \leq w} \{D[w - w_i] + c_i\}.$$

Дин. прог. снизу вверх

Дин. прог. снизу вверх

Время работы: O(nW).

Пример: W = 1030 руб. 14 руб. 16 руб. 9 руб. 1 2 3 4 5 6 7 8 9 10 0 0 0 0 0 0 0 0 0 0 0

Пример: W = 1030 руб. 14 руб. 16 руб. 9 руб. 1 2 3 4 5 6 7 8 9 10 0 0 0 0 0 0 0 0 0

Пример: W = 1030 руб. 14 руб. 16 руб. 9 руб. 1 2 3 4 5 6 7 8 9 10 9 0 0 0 0 0 0 0 0

Пример: W = 1030 руб. 14 руб. 16 руб. 9 руб. 0 1 2 3 4 5 6 7 8 9 10 9 0 0 0 0 0 0 0 0

Пример: W = 1030 руб. 14 руб. 16 руб. 9 руб. 1 2 3 4 5 6 7 8 9 10 0 0 9 14 0 0 0 0 0 0 0

Пример: W = 1030 руб. 14 руб. 16 руб. 9 руб. 0 1 2 3 4 5 6 7 8 9 10 9 14 0 0 0 0 0 0 0

Пример: W = 1030 руб. 14 руб. 16 руб. 9 руб. 1 2 3 4 5 6 7 8 9 10 0 0 9 14 18 0 0 0 0 0 0

Пример: W = 1030 руб. 14 руб. 16 руб. 9 руб. 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 $0 \le w \le W$ и $0 \le i \le n$, D[w,i] максимальная стоимость рюкзака вместимости w, если разрешено использовать только предметы $1, \ldots, i$.
- Предмет *і* либо используется, либо нет:

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

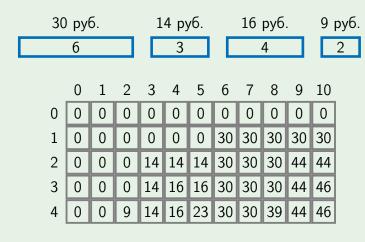
KNAPSACKWITHOUTREPSBU $(W, w_1, \ldots, w_n, c_1, \ldots, c_n)$

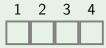
```
создать массив D[0...W,0...n]
для w от 0 до W:
  D[w,0] \leftarrow 0
для i от 0 до n:
  D[0,i] \leftarrow 0
для i от 1 до n:
  для w от 1 до W:
    D[w,i] \leftarrow D[w,i-1]
    если w_i < w:
       D[w, i] = \max(D[w, i], D[w - w_i, i - 1] + c_i)
вернуть D[W, n]
```

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

Время работы: O(nW).

вернуть D[W, n]





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

- 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.

- 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.

- 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 деляющемся на 100.

Дин. прог. сверху вниз для рюкзака с повторениями

KNAPSACKTD(w)

```
если w нет в хеш-таблице H: v \leftarrow 0 для всех i от 1 до n: если w_i \leq w: v \leftarrow \max\{v, \texttt{KnapsackTD}(w-w_i)+c_i\} H[w] \leftarrow 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 = 71345970345617824751

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