Дискретная задача о рюкзаке

Иван Сергеевич Казменко

Санкт-Петербургский Государственный Университет,
Факультет Математики и Компьютерных Наук

Пятница, 20 февраля 2026 года

Содержание

Содержание

Дискретная задача о рюкзаке

Входные данные:

- Есть n вещей и рюкзак вместимостью s
- Вещь с номером i характеризуется размером (весом) w_i и ценой c_i
- Цен нет, нужно набрать как можно больший вес ($c_i = w_i$)

Нужно выбрать некоторое подмножество вещей так, чтобы:

- Суммарный размер выбранных вещей не превосходил s
- Суммарная цена выбранных вещей была как можно больше

Дополнительное условие: w_i — положительные целые числа.

Содержание

Варианты постановки задачи

- Нужно набрать как можно больше вещей ($c_i = 1$)
  Решается жадным алгоритмом: отсортируем вещи по весу и будем брать, начиная с самой маленькой, пока они помещаются
- Цен нет, нужно набрать как можно больший вес ($c_i = w_i$)
  Решается аналогично исходной постановке
- Вещей каждого типа не одна, а сколько угодно
  Решается аналогично исходной постановке

Варианты постановки задачи

- Нужно набрать как можно больше вещей ($c_i = 1$)
  Решается жадным алгоритмом: отсортируем вещи по весу и будем брать, начиная с самой маленькой, пока они помещаются
- Цен нет, нужно набрать как можно больший вес ($c_i = w_i$)
  Решается аналогично исходной постановке
- Вещей каждого типа не одна, а сколько угодно
  Решается аналогично исходной постановке

Варианты постановки задачи

- Нужно набрать как можно больше вещей ($c_i = 1$)
  Решается жадным алгоритмом: отсортируем вещи по весу и будем брать, начиная с самой маленькой, пока они помещаются
- Цен нет, нужно набрать как можно больший вес ($c_i = w_i$)
  Решается аналогично исходной постановке
- Вещей каждого типа не одна, а сколько угодно
  Решается аналогично исходной постановке

Варианты постановки задачи

- Нужно набрать как можно больше вещей ($c_i = 1$)
  Решается жадным алгоритмом: отсортируем вещи по весу и будем брать, начиная с самой маленькой, пока они помещаются
- Цен нет, нужно набрать как можно больший вес ($c_i = w_i$)
  Решается аналогично исходной постановке
- Вещей каждого типа не одна, а сколько угодно
  Решается аналогично исходной постановке

Варианты постановки задачи

- Нужно набрать как можно больше вещей ($c_i = 1$)
  Решается жадным алгоритмом: отсортируем вещи по весу и будем брать, начиная с самой маленькой, пока они помещаются
- Цен нет, нужно набрать как можно больший вес ($c_i = w_i$)
  Решается аналогично исходной постановке
- Вещей каждого типа не одна, а сколько угодно
  Решается аналогично исходной постановке

Варианты постановки задачи

- Нужно набрать как можно больше вещей ($c_i = 1$)
  Решается жадным алгоритмом: отсортируем вещи по весу и будем брать, начиная с самой маленькой, пока они помещаются
- Цен нет, нужно набрать как можно больший вес ($c_i = w_i$)
  Решается аналогично исходной постановке
- Вещей каждого типа не одна, а сколько угодно
  Решается аналогично исходной постановке

Содержание

Пример

$$n = 4$$

$$s = 6$$

i	1	2	3	4
w_i	1	2	3	4
c_i	2	3	5	7

Пример

$$n = 4$$

$$s = 6$$

i	1	2	3	4
w_i	1	2	3	4
c_i	2	3	5	7

Оптимальное решение:

Выберем вещи с номерами 2 и 4.

Суммарный вес равен $2 + 4 = 6$.

Суммарная цена равна $3 + 7 = 10$.

Пример

$$n = 4$$
$$s = 6$$

i	1	2	3	4
w_i	1	2	3	4
c_i	2	3	5	7

Ещё одно оптимальное решение:

Выберем вещи с номерами 1, 2 и 3.

Суммарный вес равен $1 + 2 + 3 = 6$.

Суммарная цена равна $2 + 3 + 5 = 10$.

Пример

$$n = 4$$
$$s = 6$$

i	1	2	3	4
w_i	1	2	3	4
c_i	2	3	5	7

Всего есть $2 \cdot 2 \cdot 2 \cdot 2 = 16$ вариантов решения: каждую вещь можно независимо от других либо взять, либо не взять.

Пример

$$n = 4$$

$$s = 6$$

i	1	2	3	4
w_i	1	2	3	4
c_i	2	3	5	7

Некоторые решения неоптимальны:

Выберем вещи с номерами 1 и 4.

Суммарный вес равен $1 + 4 = 5$.

Суммарная цена равна $2 + 7 = 9$.

Заметим, что в этом решении не добавить ещё одну вещь.

Пример

$$n = 4$$

$$s = 6$$

i	1	2	3	4
w_i	1	2	3	4
c_i	2	3	5	7

Некоторые решения невозможны:

Выберем вещи с номерами 3 и 4.

Суммарный вес равен $3 + 4 = 7 > s$.

Содержание

Наивный алгоритм

- Переберём все возможные подмножества вещей
- Для каждого подмножества проверим, что суммарный вес не превосходит s
- Из подмножеств, прошедших проверку, выберем подмножество с максимальной суммарной ценой

Трудоёмкость: $O(2^n \cdot n)$, при аккуратной реализации $O(2^n)$.
Недостаток: большое (экспоненциальное) время работы.

Наивный алгоритм

- Переберём все возможные подмножества вещей
- Для каждого подмножества проверим, что суммарный вес не превосходит s
- Из подмножеств, прошедших проверку, выберем подмножество с максимальной суммарной ценой

Трудоёмкость: $O(2^n \cdot n)$, при аккуратной реализации $O(2^n)$.

Недостаток: большое (экспоненциальное) время работы.

Наивный алгоритм

- Переберём все возможные подмножества вещей
- Для каждого подмножества проверим, что суммарный вес не превосходит s
- Из подмножеств, прошедших проверку, выберем подмножество с максимальной суммарной ценой

Трудоёмкость: $O(2^n \cdot n)$, при аккуратной реализации $O(2^n)$.
Недостаток: большое (экспоненциальное) время работы.

Содержание

Жадные алгоритмы

- Упорядочим вещи по какому-то критерию
- Рассмотрим вещи в полученном порядке
- Возьмём те из них, на которые хватает места

Трудоёмкость: $O(n \log n)$ на сортировку $+O(n)$ на выбор.
Недостаток: жадные алгоритмы решения этой задачи неверны.

Жадные алгоритмы

- Упорядочим вещи по какому-то критерию
- Рассмотрим вещи в полученном порядке
- Возьмём те из них, на которые хватает места

Трудоёмкость: $O(n \log n)$ на сортировку $+O(n)$ на выбор.

Недостаток: жадные алгоритмы решения этой задачи неверны.

Жадные алгоритмы

- Упорядочим вещи по какому-то критерию
- Рассмотрим вещи в полученном порядке
- Возьмём те из них, на которые хватает места

Трудоёмкость: $O(n \log n)$ на сортировку $+O(n)$ на выбор.
Недостаток: жадные алгоритмы решения этой задачи неверны.

Жадные алгоритмы

1. Порядок сортировки: по убыванию цены c_i

$$n = 4$$
$$s = 6$$

i	1	2	3	4
w_i	4	3	2	1
c_i	7	5	3	2

Жадные алгоритмы

1. Порядок сортировки: по убыванию цены c_i

$$n = 4$$
$$s = 6$$

i	1	2	3	4
w_i	4	3	2	1
c_i	7	5	3	2

Найденное решение:

Выберем вещи с номерами 1 и 3.

Суммарный вес равен $4 + 2 = 6$.

Суммарная цена равна $7 + 3 = 10$.

Решение оптимально.

Жадные алгоритмы

1. Порядок сортировки: по убыванию цены c_i

$$n = 3$$
$$s = 3$$

i	1	2	3
w_i	3	2	1
c_i	4	3	2

Жадные алгоритмы

1. Порядок сортировки: по убыванию цены c_i

$$n = 3$$
$$s = 3$$

i	1	2	3
w_i	3	2	1
c_i	4	3	2

Найденное решение:
Выберем вещь с номером 1.
Суммарный вес равен 3.
Суммарная цена равна 4.
Но это решение неоптимально.

Жадные алгоритмы

1. Порядок сортировки: по убыванию цены c_i

$$n = 3$$
$$s = 3$$

i	1	2	3
w_i	3	2	1
c_i	4	3	2

Оптимальное решение:
Выберем вещи с номерами 2 и 3.
Суммарный вес равен $2 + 1 = 3$.
Суммарная цена равна $3 + 2 = 5$.
Значит, алгоритм работает неверно.

Жадные алгоритмы

2. Порядок сортировки: по убыванию веса w_i

$$n = 4$$
$$s = 6$$

i	1	2	3	4
w_i	4	3	2	1
c_i	7	5	3	2

Жадные алгоритмы

2. Порядок сортировки: по убыванию веса w_i

$$n = 4$$
$$s = 6$$

i	1	2	3	4
w_i	4	3	2	1
c_i	7	5	3	2

Найденное решение:

Выберем вещи с номерами 1 и 3.

Суммарный вес равен $4 + 2 = 6$.

Суммарная цена равна $7 + 3 = 10$.

Решение оптимально.

Жадные алгоритмы

2. Порядок сортировки: по убыванию веса w_i

$$n = 3$$
$$s = 3$$

i	1	2	3
w_i	3	2	1
c_i	4	3	2

Жадные алгоритмы

2. Порядок сортировки: по убыванию веса w_i

$$n = 3$$
$$s = 3$$

i	1	2	3
w_i	3	2	1
c_i	4	3	2

Найденное решение:
Выберем вещь с номером 1.
Суммарный вес равен 3.
Суммарная цена равна 4.
Но это решение неоптимально.

Жадные алгоритмы

2. Порядок сортировки: по убыванию веса w_i

$$n = 3$$
$$s = 3$$

i	1	2	3
w_i	3	2	1
c_i	4	3	2

Оптимальное решение:
Выберем вещи с номерами 2 и 3.
Суммарный вес равен $2 + 1 = 3$.
Суммарная цена равна $3 + 2 = 5 > 4$.
Значит, алгоритм работает неверно.

Жадные алгоритмы

3. Порядок сортировки: по возрастанию веса w_i

$$n = 4$$
$$s = 6$$

i	1	2	3	4
w_i	1	2	3	4
c_i	2	3	5	7

Жадные алгоритмы

3. Порядок сортировки: по возрастанию веса w_i

$$n = 4$$
$$s = 6$$

i	1	2	3	4
w_i	1	2	3	4
c_i	2	3	5	7

Найденное решение:

Выберем вещи с номерами 1, 2 и 3.

Суммарный вес равен $1 + 2 + 3 = 6$.

Суммарная цена равна $2 + 3 + 5 = 10$.

Решение оптимально.

Жадные алгоритмы

3. Порядок сортировки: по возрастанию веса w_i

$$n = 3$$
$$s = 5$$

i	1	2	3
w_i	2	3	4
c_i	1	2	4

Жадные алгоритмы

3. Порядок сортировки: по возрастанию веса w_i

$$n = 3$$
$$s = 5$$

i	1	2	3
w_i	2	3	4
c_i	1	2	4

Найденное решение:
Выберем вещи с номерами 1 и 2.
Суммарный вес равен $2 + 3 = 5$.
Суммарная цена равна $1 + 2 = 3$.
Но это решение неоптимально.

Жадные алгоритмы

3. Порядок сортировки: по возрастанию веса w_i

$$n = 3$$
$$s = 5$$

i	1	2	3
w_i	2	3	4
c_i	1	2	4

Оптимальное решение:
Выберем вещь с номером 3.
Суммарный вес равен 4.
Суммарная цена равна $4 > 3$.
Значит, алгоритм работает неверно.

Жадные алгоритмы

4. Порядок сортировки: по убыванию «удельной стоимости» $\frac{c_i}{w_i}$

$$n = 4$$
$$s = 6$$

i	1	2	3	4
w_i	1	4	3	2
c_i	2	7	5	3
$\frac{c_i}{w_i}$	2.0	1.75	1.666 . . .	1.5

Жадные алгоритмы

4. Порядок сортировки: по убыванию «удельной стоимости» $\frac{c_i}{w_i}$

$$n = 4$$
$$s = 6$$

i	1	2	3	4
w_i	1	4	3	2
c_i	2	7	5	3
$\frac{c_i}{w_i}$	2.0	1.75	1.666 . . .	1.5

Найденное решение:

Выберем вещи с номерами 1 и 2.

Суммарный вес равен $1 + 4 = 5$.

Суммарная цена равна $2 + 7 = 9$.

Но это решение неоптимально $(9 < 10)$.

Значит, алгоритм работает неверно.

Содержание

Решение динамическим программированием

- Воспользуемся тем, что веса — небольшие целые числа
- Будем рассматривать вещи по порядку

Подзадача:

- Пусть мы рассмотрели первые k вещей
- Для каждого целого суммарного веса u $(0 \leqslant u \leqslant s)$ выясним, какую максимальную суммарную цену могут иметь вещи с таким весом

Решение динамическим программированием

- Воспользуемся тем, что веса — небольшие целые числа
- Будем рассматривать вещи по порядку

Подзадача:

- Пусть мы рассмотрели первые k вещей
- Для каждого целого суммарного веса u $(0 \leqslant u \leqslant s)$ выясним, какую максимальную суммарную цену могут иметь вещи с таким весом

Решение динамическим программированием

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.

Целевая функция: $f(k, u)$ — максимальная суммарная цена.

База: $f(0, 0) = 0$, $f(0, u) = -\infty$ для всех $u > 0$.

Ответ: Максимум $f(n, u)$ по всем $0 \leqslant u \leqslant s$.

Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.

Решение динамическим программированием

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.
Целевая функция: $f(k, u)$ — максимальная суммарная цена.
База: $f(0, 0) = 0$, $f(0, u) = -\infty$ для всех $u > 0$.
Ответ: Максимум $f(n, u)$ по всем $0 \leqslant u \leqslant s$.
Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.
Динамика вперёд: из состояния $(k-1, u)$ есть два перехода.

- В (k, u), если не брать вещь k, цена не изменилась
- В $(k, u + w_k)$, если брать вещь k, цена выросла на c_k

```
for k := 1 upto n:
  for u := 0 upto s:
    if f[k][u] < f[k - 1][u]:
      f[k][u] := f[k - 1][u]
    if f[k][u + w[k]] < f[k - 1][u] + c[k]:
      f[k][u + w[k]] := f[k - 1][u] + c[k]
```

Решение динамическим программированием

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.
Целевая функция: $f(k, u)$ — максимальная суммарная цена.
База: $f(0, 0) = 0$, $f(0, u) = -\infty$ для всех $u > 0$.
Ответ: Максимум $f(n, u)$ по всем $0 \leqslant u \leqslant s$.
Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.
Динамика вперёд: из состояния $(k-1, u)$ есть два перехода.

- В (k, u), если не брать вещь k, цена не изменилась
- В $(k, u + w_k)$, если брать вещь k, цена выросла на c_k

```
for k := 1 upto n:
  for u := 0 upto s:
    if f[k][u] < f[k - 1][u]:
      f[k][u] := f[k - 1][u]
    if u + w[k] <= s:
      if f[k][u + w[k]] < f[k - 1][u] + c[k]:
        f[k][u + w[k]] := f[k - 1][u] + c[k]
```

Решение динамическим программированием

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.
Целевая функция: $f(k, u)$ — максимальная суммарная цена.
База: $f(0, 0) = 0$, $f(0, u) = -\infty$ для всех $u > 0$.
Ответ: Максимум $f(n, u)$ по всем $0 \leqslant u \leqslant s$.
Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.
Динамика назад: в состояние (k, u) есть два перехода.

- Из $(k-1, u)$, если не брать вещь k, цена не изменилась
- Из $(k-1, u-w_k)$, если брать вещь k, цена выросла на c_k

```
for k := 1 upto n:
  for u := 0 upto s:
    if f[k][u] < f[k - 1][u]:
      f[k][u] := f[k - 1][u]
    if f[k][u] < f[k - 1][u - w[k]] + c[k]:
      f[k][u] := f[k - 1][u - w[k]] + c[k]
```

Решение динамическим программированием

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.
Целевая функция: $f(k, u)$ — максимальная суммарная цена.
База: $f(0, 0) = 0$, $f(0, u) = -\infty$ для всех $u > 0$.
Ответ: Максимум $f(n, u)$ по всем $0 \leqslant u \leqslant s$.
Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.
Динамика назад: в состояние (k, u) есть два перехода.

- Из $(k-1, u)$, если не брать вещь k, цена не изменилась
- Из $(k-1, u-w_k)$, если брать вещь k, цена выросла на c_k

```
for k := 1 upto n:
  for u := 0 upto s:
    if f[k][u] < f[k - 1][u]:
      f[k][u] := f[k - 1][u]
    if u - w[k] >= 0:
      if f[k][u] < f[k - 1][u - w[k]] + c[k]:
        f[k][u] := f[k - 1][u - w[k]] + c[k]
```

Решение динамическим программированием

Анализ:

- Время работы: $O(n \cdot s)$
- Требуемая память: $O(n \cdot s)$

```
for k := 1 upto n:
  for u := 0 upto s:
    if f[k][u] < f[k - 1][u]:
      f[k][u] := f[k - 1][u]
    if f[k][u] < f[k - 1][u - w[k]] + c[k]:
      f[k][u] := f[k - 1][u - w[k]] + c[k]
```

Решение динамическим программированием

Оптимизация по памяти: заметим, что $f(k, u) \geqslant f(k-1, u)$.

- Время работы: $O(n \cdot s)$
- Требуемая память: $O(s)$

```
for k := 1 upto n:
  for u := 0 upto s:
    if f[k][u] < f[k - 1][u]:
      f[k][u] := f[k - 1][u]
    if f[k][u] < f[k - 1][u - w[k]] + c[k]:
      f[k][u] := f[k - 1][u - w[k]] + c[k]
```

Избавимся от размерности k в массиве.

Решение динамическим программированием

Оптимизация по памяти: заметим, что $f(k, u) \geqslant f(k-1, u)$.

- Время работы: $O(n \cdot s)$
- Требуемая память: $O(s)$

```
for k := 1 upto n:
  for u := 0 upto s:
    if f[u] < f[u]:
      f[u] := f[u]
    if f[u] < f[u - w[k]] + c[k]:
      f[u] := f[u - w[k]] + c[k]
```

Первый переход теперь делается автоматически.

Решение динамическим программированием

Оптимизация по памяти: заметим, что $f(k, u) \geqslant f(k-1, u)$.

- Время работы: $O(n \cdot s)$
- Требуемая память: $O(s)$

```
for k := 1 upto n:
  for u := w[k] upto s:


    if f[u] < f[u - w[k]] + c[k]:
      f[u] := f[u - w[k]] + c[k]
```

Решение динамическим программированием

Оптимизация по памяти: заметим, что $f(k, u) \geqslant f(k-1, u)$.

- Время работы: $O(n \cdot s)$
- Требуемая память: $O(s)$

```
for k := 1 upto n:
  for u := w[k] upto s:
    if f[u] < f[u - w[k]] + c[k]:
      f[u] := f[u - w[k]] + c[k]
```

Ошибка: вещи могут быть взяты более одного раза.
Зато получилось решение одного из вариантов постановки задачи.

Решение динамическим программированием

Оптимизация по памяти: заметим, что $f(k, u) \geqslant f(k-1, u)$.

- Время работы: $O(n \cdot s)$
- Требуемая память: $O(s)$

```
for k := 1 upto n:
  for u := s downto w[k]:
    if f[u] < f[u - w[k]] + c[k]:
      f[u] := f[u - w[k]] + c[k]
```

Теперь каждая вещь берётся не более одного раза.

Содержание

Восстановление решения

Нужно узнать не только ответ, но и как он был получен.

Преимущества:

- Решение добавляет ответу наглядности
- Проще отлаживать программу
- Существует общий метод восстановления решения

Недостатки:

- Дополнительный объём кода
- Некоторые оптимизации становятся невозможными

Восстановление решения

Нужно узнать не только ответ, но и как он был получен.

Преимущества:

- Решение добавляет ответу наглядности
- Проще отлаживать программу
- Существует общий метод восстановления решения

Недостатки:

- Дополнительный объём кода
- Некоторые оптимизации становятся невозможными

Восстановление решения

Нужно узнать не только ответ, но и как он был получен.

Преимущества:

- Решение добавляет ответу наглядности
- Проще отлаживать программу
- Существует общий метод восстановления решения

Недостатки:

- Дополнительный объём кода
- Некоторые оптимизации становятся невозможными

Восстановление решения

Нужно узнать не только ответ, но и как он был получен.

```
for k := 1 upto n:
  for u := 0 upto s:
    if f[k][u] < f[k - 1][u]:
      f[k][u] := f[k - 1][u]
    if f[k][u] < f[k - 1][u - w[k]] + c[k]:
      f[k][u] := f[k - 1][u - w[k]] + c[k]
```

Заведём дополнительный массив p, в котором для каждого состояния запишем, откуда мы в него пришли.

Восстановление решения

Нужно узнать не только ответ, но и как он был получен.

```
for k := 1 upto n:
  for u := 0 upto s:
    if f[k][u] < f[k - 1][u]:
      f[k][u] := f[k - 1][u]
      p[k][u] := 0
    if f[k][u] < f[k - 1][u - w[k]] + c[k]:
      f[k][u] := f[k - 1][u - w[k]] + c[k]
      p[k][u] := 1
```

Восстановление решения

Нужно узнать не только ответ, но и как он был получен.
Восстанавливаем решение:

- Сначала находим ответ
- Затем идём с конца, каждый раз находя предыдущее состояние с помощью массива p

```
k := n
u := 0
for v := 1 upto s:
  if f[k][u] < f[k][v]:
    u := v

while k > 0:
  if p[k][u] == 1:
    output k
    u := u - w[k]
  k := k - 1
```

Восстановление решения

Нужно узнать не только ответ, но и как он был получен.
Восстанавливаем решение:

- Сначала находим ответ
- Затем идём с конца, каждый раз находя предыдущее состояние с помощью массива p

```
k := n
u := 0
for v := 1 upto s:
  if f[k][u] < f[k][v]:
    u := v

while k > 0:
  if p[k][u] == 1:
    output k
    u := u - w[k]
  k := k - 1
```

Содержание

Случай $c_i = w_i$

- Есть n вещей и рюкзак вместимостью s
- Вещь с номером i характеризуется размером (весом) w_i
- Цен нет, нужно набрать как можно больший вес ($c_i = w_i$)

Упростим наше решение:

- Воспользуемся тем, что веса — небольшие целые числа
- Будем рассматривать вещи по порядку

Подзадача:

- Пусть мы рассмотрели первые k вещей
- Для каждого целого суммарного веса u ($0 \leqslant u \leqslant s$) выясним, можно ли набрать ровно такой вес

Случай $c_i = w_i$

- Есть n вещей и рюкзак вместимостью s
- Вещь с номером i характеризуется размером (весом) w_i
- Цен нет, нужно набрать как можно больший вес ($c_i = w_i$)

Упростим наше решение:

- Воспользуемся тем, что веса — небольшие целые числа
- Будем рассматривать вещи по порядку

Подзадача:

- Пусть мы рассмотрели первые k вещей
- Для каждого целого суммарного веса u ($0 \leqslant u \leqslant s$) выясним, можно ли набрать ровно такой вес

Случай $c_i = w_i$

- Есть n вещей и рюкзак вместимостью s
- Вещь с номером i характеризуется размером (весом) w_i
- Цен нет, нужно набрать как можно больший вес ($c_i = w_i$)

Упростим наше решение:

- Воспользуемся тем, что веса — небольшие целые числа
- Будем рассматривать вещи по порядку

Подзадача:

- Пусть мы рассмотрели первые k вещей
- Для каждого целого суммарного веса u ($0 \leqslant u \leqslant s$) выясним, можно ли набрать ровно такой вес

Случай $c_i = w_i$

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.
Целевая функция: $f(k, u)$ — можно ли набрать такой вес.
База: $f(0, 0) = \texttt{true}$, $f(0, u) = \texttt{false}$ для всех $u > 0$.
Ответ: Максимальное u, для которого $f(n, u) = \texttt{true}$.
Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.

Случай $c_i = w_i$

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.
Целевая функция: $f(k, u)$ — можно ли набрать такой вес.
База: $f(0, 0) = \texttt{true}$, $f(0, u) = \texttt{false}$ для всех $u > 0$.
Ответ: Максимальное u, для которого $f(n, u) = \texttt{true}$.
Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.
Динамика вперёд:

```
for k := 1 upto n:
  for u := 0 upto s:
    f[k][u] := f[k][u] or f[k - 1][u]
    if u + w[k] <= s:
      f[k][u + w[k]] := f[k][u + w[k]] or
                        f[k - 1][u]
```

Случай $c_i = w_i$

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.
Целевая функция: $f(k, u)$ — можно ли набрать такой вес.
База: $f(0, 0) = \texttt{true}$, $f(0, u) = \texttt{false}$ для всех $u > 0$.
Ответ: Максимальное u, для которого $f(n, u) = \texttt{true}$.
Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.
Динамика назад:

```
for k := 1 upto n:
  for u := 0 upto s:
    f[k][u] := f[k - 1][u]
    if u - w[k] >= 0:
      f[k][u] := f[k][u] or f[k - 1][u - w[k]]
```

Случай $c_i = w_i$

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.
Целевая функция: $f(k, u)$ — можно ли набрать такой вес.
База: $f(0, 0) = \texttt{true}$, $f(0, u) = \texttt{false}$ для всех $u > 0$.
Ответ: Максимальное u, для которого $f(n, u) = \texttt{true}$.
Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.
Избавимся от размерности k в массиве:

```
for k := 1 upto n:
  for u := 0 upto s:
    f[u] := f[u]
    if u - w[k] >= 0:
      f[u] := f[u] or f[u - w[k]]
```

Случай $c_i = w_i$

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.
Целевая функция: $f(k, u)$ — можно ли набрать такой вес.
База: $f(0, 0) = \texttt{true}$, $f(0, u) = \texttt{false}$ для всех $u > 0$.
Ответ: Максимальное u, для которого $f(n, u) = \texttt{true}$.
Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.
Уберём переход, который теперь не нужен:

```
for k := 1 upto n:
  for u := w[k] upto s:
    f[u] := f[u] or f[u - w[k]]
```

Случай $c_i = w_i$

Состояние: (k, u) — количество рассмотренных вещей и суммарный вес.
Целевая функция: $f(k, u)$ — можно ли набрать такой вес.
База: $f(0, 0) = \texttt{true}$, $f(0, u) = \texttt{false}$ для всех $u > 0$.
Ответ: Максимальное u, для которого $f(n, u) = \texttt{true}$.
Переход: пусть известны все $f(k-1, u)$, получим все $f(k, u)$.
Развернём цикл, чтобы не брать вещи много раз:

```
for k := 1 upto n:
  for u := s downto w[k]:
    f[u] := f[u] or f[u - w[k]]
```

Случай $c_i = w_i$

Как теперь сделать восстановление ответа?

Заметим, что за время работы алгоритма каждое $f(u)$ меняет значение с `false` на `true` не больше одного раза.

Давайте записывать не логическое значение, а момент этого изменения!

Случай $c_i = w_i$

Как теперь сделать восстановление ответа?
Заметим, что за время работы алгоритма каждое $f(u)$ меняет значение с `false` на `true` не больше одного раза.
Давайте записывать не логическое значение, а момент этого изменения!
Состояние: (u) — суммарный вес.

Целевая функция: $g(u)$ — минимальное количество вещей k, которые нужно *рассмотреть с начала*,
чтобы набрать такой вес.
База: $g(0) = 0$.
Изначально присвоим $g(u) = +\infty$ для всех $u > 0$.
Ответ: Максимальное u, для которого $g(u) < +\infty$.
Переход: рассмотрим k-ю вещь.

Случай $c_i = w_i$

Состояние: (u) — суммарный вес.
Целевая функция: $g(u)$ — минимальное количество вещей k, которые нужно *рассмотреть с начала*, чтобы набрать такой вес.
База: $g(0) = 0$.
Изначально присвоим $g(u) = +\infty$ для всех $u > 0$.
Ответ: Максимальное u, для которого $g(u) < +\infty$.
Переход: рассмотрим k-ю вещь.

```
for k := 1 upto n:
  for u := w[k] to s:
    if (g[u] = +inf) and (g[u - w[k]] < k):
      g[u] := k
```

Случай $c_i = w_i$

Восстановление ответа: $g(u)$ — номер вещи, которую нужно взять и перейти к подзадаче.

Случай $c_i = w_i$

Восстановление ответа: $g(u)$ — номер вещи, которую нужно взять и перейти к подзадаче.

```
u := s
while g[u] = +inf:
  u := u - 1

while u > 0:
  k := g[u]
  output k
  u := u - w[k]
```

Случай $c_i = w_i$

Восстановление ответа: $g(u)$ — номер вещи, которую нужно взять и перейти к подзадаче.

```
u := s
while g[u] = +inf:
  u := u - 1

while u > 0:
  k := g[u]
  output k
  u := u - w[k]
```

Анализ:

- Время работы: $O(n \cdot s)$
- Требуемая память: $O(s)$

Содержание

Две кучки монет

Задача:

- Есть n монет
- Монета с номером i имеет стоимость c_i
- Нужно разделить монеты на две кучки как можно более честно: так, чтобы разность между суммами монет в двух кучках была как можно меньше

Будем решать так:

- Рассматриваем монеты по порядку
- Каждую монету либо берём в первую кучку, либо во вторую кучку
- Считаем рюкзаком только первую кучку
- Сумма во второй кучке однозначно восстанавливается по сумме в первой кучке и общей сумме монет

Две кучки монет

Задача:

- Есть n монет
- Монета с номером i имеет стоимость c_i
- Нужно разделить монеты на две кучки как можно более честно: так, чтобы разность между суммами монет в двух кучках была как можно меньше

Будем решать так:

- Рассматриваем монеты по порядку
- Каждую монету либо берём в первую кучку, либо во вторую кучку
- Считаем рюкзаком только первую кучку
- Сумма во второй кучке однозначно восстанавливается по сумме в первой кучке и общей сумме монет

Две кучки монет

Будем решать так:

- Рассматриваем монеты по порядку
- Каждую монету либо берём в первую кучку, либо во вторую кучку
- Считаем рюкзаком только первую кучку
- Сумма во второй кучке однозначно восстанавливается по сумме в первой кучке и общей сумме монет

В конце найдём достижимую сумму, наиболее близкую к половине общей суммы.
Получилось свести задачу к задаче о рюкзаке в случае $c_i = w_i$.

Вопросы?

Вопросы?