Задача о рюкзаке

Задача:

Задача о рюкзаке *(англ. Knapsack problem) — дано N предметов, n_i предмет имеет массу $w_i > 0$ и стоимость $p_i > 0$. Необходимо выбрать из этих предметов такой набор, чтобы суммарная масса не превосходила заданной величины W (вместимость рюкзака), а суммарная стоимость была максимальна.*

Содержание [убрать]

Формулировка задачи

Дано N предметов, W — вместимость рюкзака, $w = \{w_1, w_2, \dots, w_N\}$ — соответствующий ему набор положительных целых весов, $p = \{p_1, p_2, \dots, p_N\}$ — соответствующий ему набор положительных целых стоимостей. Нужно найти набор бинарных величин $B = \{b_1, b_2, \dots, b_N\}$, где $b_i = 1$, если предмет n_i включен в набор, $b_i = 0$, если предмет n_i не включен, и такой что:

1. $b_1 w_1 + \dots + b_N w_N \leqslant W$
2. $b_1 p_1 + \dots + b_N p_N$ максимальна.

Варианты решения

Задачу о рюкзаке можно решить несколькими способами:

- Перебирать все подмножества набора из N предметов. Сложность такого решения $O(2^N)$.
- Методом Meet-in-the-middle. Сложность решения $O(2^{N/2}N)$
- Метод динамического программирования. Сложность — $O(NW)$.

Метод динамического программирования

Пусть $A(k, s)$ есть максимальная стоимость предметов, которые можно уложить в рюкзак вместимости s, если можно использовать только первые k предметов, то есть $\{n_1, n_2, \dots, n_k\}$, назовем этот набор допустимых предметов для $A(k, s)$.

$A(k, 0) = 0$

$A(0, s) = 0$

Найдем $A(k, s)$. Возможны 2 варианта:

1. Если предмет k не попал в рюкзак. Тогда $A(k, s)$ равно максимальной стоимости рюкзака с такой же вместимостью и набором допустимых предметов $\{n_1, n_2, \dots, n_{k-1}\}$, то есть $A(k, s) = A(k-1, s)$
2. Если k попал в рюкзак. Тогда $A(k, s)$ равно максимальной стоимости рюкзака, где вес s уменьшаем на вес k-ого предмета и набор допустимых предметов $\{n_1, n_2, \dots, n_{k-1}\}$ плюс стоимость k, то есть $A(k-1, s-w_k) + p_k$

$$A(k, s) = \begin{cases} A(k-1, s), & b_k = 0 \\ A(k-1, s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k, s) = \max(A(k-1, s), A(k-1, s-w_k) + p_k)$

Стоимость искомого набора равна $A(N, W)$, так как нужно найти максимальную стоимость рюкзака, где все предметы допустимы и вместимость рюкзака W.

Восстановим набор предметов, входящих в рюкзак

Будем определять, входит ли n_i предмет в искомый набор. Начинаем с элемента $A(i, w)$, где $i = N$, $w = W$. Для этого сравниваем $A(i, w)$ со следующими значениями:

1. Максимальная стоимость рюкзака с такой же вместимостью и набором допустимых предметов $\{n_1, n_2, \dots, n_{i-1}\}$, то есть $A(i-1, w)$
2. Максимальная стоимость рюкзака с вместимостью на w_i меньше и набором допустимых предметов $\{n_1, n_2, \dots, n_{i-1}\}$ плюс стоимость p_i, то есть $A(i-1, w-w_i) + p_i$

Заметим, что при построении A мы выбирали максимум из этих значений и записывали в $A(i, w)$. Тогда будем сравнивать $A(i, w)$ с $A(i-1, w)$, если равны, тогда n_i не входит в искомый набор, иначе входит.

Метод динамического программирование всё равно не позволяет решать задачу за полиномиальное время, потому что его сложность зависит от максимального веса. Задача о ранце (или задача о рюкзаке) — одна из NP-полных задач комбинаторной оптимизации.

Реализация

Сначала генерируем A.

```
for i = 0 to w
  A[0][i] = 0
for i = 0 to n
  A[i][0] = 0                                     //Первые элементы приравниваем к 0
for k = 1 to n
  for s = 1 to w                                  //Перебираем для каждого k все вместимости
    if s >= w[k]                                  //Если текущий предмет вмещается в рюкзак
      A[k][s] = max(A[k - 1][s], A[k - 1][s - w[k]] + p[k]) //Выбираем класть его или нет
    else
      A[k][s] = A[k - 1][s]                       //Иначе, не кладем
```

Затем найдем набор ans предметов, входящих в рюкзак, рекурсивной функцией:

```
function findAns(int k, int s)
  if A[k][s] == 0
    return
  if A[k - 1][s] == A[k][s]
    findAns(k - 1, s)
  else
    findAns(k - 1, s - w[k])
    ans.push(k)
```

Сложность алгоритма $O(NW)$

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

Числа от 0 до 13 в первой строчке обозначают вместимость рюкзака.

В первой строке как только вместимость рюкзака $n \geqslant 3$, добавляем в рюкзак 1 предмет.

Рассмотрим $k = 3$, при каждом $s \geqslant 5$(так как $w_3 = 5$) сравниваем $A[k-1][s]$ и $A[k-1][s-w_3] + p_3$ и записываем в $A[k][s]$ стоимость либо рюкзака без третьего предмета, но с таким же весом, либо с третьим предметом, тогда стоимость равна стоимости третьего предмета плюс стоимость рюкзака с вместимостью на w_3 меньше.

Максимальная стоимость рюкзака находится в $A(5, 13)$.

Восстановление набора предметов, из которых состоит максимально дорогой рюкзак.

Начиная с $A(5, 13)$ восстанавливаем ответ. Будем идти в обратном порядке по k. *Красным фоном обозначим наш путь*

	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

Таким образом, в набор входит 2 и 4 предмет.

Стоимость рюкзака: $6 + 7 = 13$

Вес рюкзака: $4 + 8 = 12$

Другие задачи семейства

Ограниченный рюкзак

Задача:

Ограниченный рюкзак *(англ. Bounded Knapsack Problem) — обобщение классической задачи, когда любой предмет может быть взят некоторое количество раз.*

Формулировка Задачи

Каждый предмет может быть выбран ограниченное b_i число раз. Задача выбрать число x_i предметов каждого типа так, чтобы

- максимизировать общую стоимость: $\sum_{i=1}^{N} p_i x_i$;
- выполнялось условие совместности: $\sum_{i=1}^{N} w_i x_i \leqslant W$;

где $x_i \in (0, 1, \dots, b_i)$ для всех $i = 1, 2, \dots, N$.

Варианты решения

При небольших b_i решается сведением к классической задаче о рюкзаке. В иных случаях:

- Методом ветвей и границ.
- Методом динамического программирования.

Метод динамического программирования

Пусть $d(i, c)$ максимальная стоимость любого возможного числа предметов типов от 1 до i, суммарным весом до c.

Заполним $d(0, c)$ нулями.

Тогда меняя i от 1 до N, рассчитаем на каждом шаге $d(i, c)$, для c от 1 до W, по рекуррентной формуле:

$d(i, c) = \max(d(i, c), d(i-1, c-lw_i) + lp_i)$ по всем целым l из промежутка $0 \leqslant l \leqslant min(b_i, \lfloor c/w_i \rfloor)$

Если не нужно восстанавливать ответ, то можно использовать одномерный массив $d(c)$ вместо двумерного.

После выполнения в $d(N, W)$ будет лежать максимальная стоимость предметов, помещающихся в рюкзак.

Реализация

```
for i = 0 to w                          //База
  d[0][i] = 0
for i = 1 to n
  for c = 1 to w                        //Перебираем для каждого i, все вместимости
    d[i][c] = d[i - 1][c]
    for l = min(b[i], c / w[i]) downto 1     //Ищем l для которого выполняется максимум
        d[i][c] =  max(d[i][c], d[i - 1][c - l * w[i]] + p[i] * l)
```

Сложность алгоритма $O(NW^2)$.

Неограниченный рюкзак

Задача:

Неограниченный рюкзак *(англ.Unbounded Knapsack Problem) — обобщение ограниченного рюкзака, в котором любой предмет может быть выбран любое количество раз.*

Формулировка Задачи

Каждый предмет может быть выбран любое число раз. Задача выбрать количество x_i предметов каждого типа так, чтобы

- максимизировать общую стоимость: $\sum_{i=1}^{N} p_i x_i$;
- выполнялось условие совместности: $\sum_{i=1}^{N} w_i x_i \leqslant W$;

где $x_i \geqslant 0$ целое, для всех $i = 1, 2, \dots, N$.

Варианты решения

Самые распространенные методы точного решения это:

- Метод ветвей и границ.
- Метод динамического программирования.

Метод динамического программирования

Пусть $d(i, c)$ - максимальная стоимость любого количества вещей типов от 1 до i, суммарным весом до c включительно.

Заполним $d(0, c)$ нулями.

Тогда меняя i от 1 до N, рассчитаем на каждом шаге $d(i, c)$, для c от 0 до W, по рекуррентной формуле:

$$d(i, c) = \begin{cases} d(i-1, c) & for\ c = 0, \dots, w_i - 1; \\ \max(d(i-1, c), d(i, c-w_i) + p_i) & for\ c = w_i, \dots, W; \end{cases}$$

После выполнения в $d(N, W)$ будет лежать максимальная стоимость предметов, помещающихся в рюкзак.

Если не нужно восстанавливать ответ, то можно использовать одномерный массив $d(c)$ вместо двумерного и использовать формулу:

$d(c) = \max(d(c), d(c-w_i) + p_i)$

Сложность алгоритма $O(NW)$.

Непрерывный рюкзак

Задача:

Непрерывный рюкзак *(англ. Continuous knapsack problem) — вариант задачи, в котором возможно брать любую дробную часть от предмета, при этом удельная стоимость сохраняется.*

Формулировка Задачи

Задача выбрать часть x_i каждого предмета так, чтобы

- максимизировать общую стоимость: $\sum_{i=1}^{N} p_i x_i$;
- выполнялось условие совместности: $\sum_{i=1}^{N} w_i x_i \leqslant W$;

где $0 \leqslant x_i \leqslant 1$ дробное, для всех $i = 1, 2, \dots, N$.

Варианты решения

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

Реализация

```
sort()                        //Сортируем в порядке убывания удельной стоимости.
```

```
for i = 1 to n                //Идем по предметам
  if w > w[i]                 //Если помещается — берем
    sum += p[i]
    w -= w[i]
  else
    sum += w / w[i] * p[i]    //Иначе берем сколько можно и выходим
    break
```

Задача о суммах подмножеств

Задача:

Задача о суммах подмножеств *(англ. Subset sum problem, Value Independent Knapsack Problem) — задача из семейства, в которой стоимость предмета совпадает с его весом.*

Формулировка Задачи

Нужно выбрать подмножество так, чтобы сумма ближе всего к W, но не превысила его. Формально, нужно найти набор бинарных величин x_i, так чтобы

- максимизировать общую стоимость: $\sum_{i=1}^{N} w_i x_i$;
- выполнялось условие совместности: $\sum_{i=1}^{N} w_i x_i \leqslant W$;

$x_j = 1$ если j предмет назначен рюкзаку, иначе $x_{ij} = 0$, для всех $i = 1, 2, \dots, N$.

Варианты решения

Для решения пригодны любые методы применяемые для классической задачи, однако специализированые алгоритмы обычно более оптимальны по параметрам. Используются:

- Метод динамического программирования.
- Использовать различные сложные алгоритмы [1][2][3][4].

Метод динамического программирования

Пусть $d(i, c)$ максимальная сумма $\leqslant c$, подмножества взятого из $1, \dots, i$ элементов.

Заполним $d(0, c)$ нулями.

Тогда меняя i от 1 до N, рассчитаем на каждом шаге $d(i, c)$, для c от 0 до W, по рекуррентной формуле:

$$d(i, c) = \begin{cases} d(i-1, c) & for\ c = 0, \dots, w_i - 1; \\ \max(d(i-1, c), d(i-1, c-w_i) + w_i) & for\ c = w_i, \dots, W; \end{cases}$$

После выполнения в $d(N, W)$ будет лежать максимальная сумма подмножества, не превышающая заданное значение.

Сложность алгоритма $O(NW)$.

Задача о размене

Задача:

Задача о размене *(англ. Change-Making problem) — имеются N неисчерпаемых типов предметов с весами w_i. Нужно наполнить рюкзак предметами с суммарным весом W.*

Часто задачу ставят как, дать сдачу наименьшим количеством монет.

Формулировка Задачи

Каждый предмет может быть выбран любое число раз. Задача выбрать количество x_i предметов каждого типа так, чтобы

- минимизировать количество взятых предметов: $\sum_{i=1}^{N} x_i$;
- сумма весов выбранных предметов равнялась вместимости рюкзака: $\sum_{i=1}^{N} w_i x_i = W$;

Где $x_i \geqslant 0$ целое, для всех $i = 1, 2, \dots, N$.

Варианты решения

Самые распространенные методы точного решения это:

- Метод ветвей и границ.
- Метод динамического программирования.

Метод динамического программирования

Пусть $d(i, c)$ минимальное число предметов, типов от 1 до i, необходимое, чтобы заполнить рюкзак вместимостью c.

Пусть $d(0, 0) = 0$, а $d(0, c) = \inf$ для всех $c > 0$.

Тогда меняя i от 1 до N, рассчитаем на каждом шаге $d(i, c)$, для c от 0 до W, по рекуррентной формуле:

$$d(i, c) = \begin{cases} d(i-1, c) & for\ c = 0, \dots, w_i - 1; \\ min(d(i-1, c), d(i, c-w_i) + 1) & for\ c = w_i, \dots, W; \end{cases}$$

После выполнения в $d(N, W)$ будет лежать максимальная стоимость предметов, помещающихся в рюкзак.

Если не нужно восстанавливать ответ, то можно использовать одномерный массив $d(c)$ вместо двумерного и использовать формулу:

$d(c) = min(d(c), d(c-w_i) + 1) \qquad for\ c = w_i, \dots, W$

Сложность алгоритма $O(NW)$.

Задача об упаковке

Задача:

Задача об упаковке *(англ. Bin Packing Problem) — имеются N рюкзаков вместимости W и столько же предметов с весами w_i. Нужно распределить все предметы, задействовав минимальное количество рюкзаков.*

Формулировка Задачи

Математически задачу можно представить так:

- минимизировать количество рюкзаков: $\sum_{i=1}^{N} y_i$;
- так чтобы выполнялось условие на совместность: $\sum_{i=1}^{N} w_i x_{ij} \leqslant W y_j \qquad j \in 1, \dots, N$;

$x_{ij} = 1$ если j предмет назначен i рюкзаку. Иначе $x_{ij} = 0$.

$y_i = 1$ если i рюкзак используется. Иначе $y_i = 0$.

Варианты решения

Применение динамического программирования нецелесообразно. Обычно применяют аппроксимационные алгоритмы, либо используют метод ветвей и границ.

Мультипликативный рюкзак

Задача:

Мультипликативный рюкзак *(англ. Multiple Knapsack Problem) — есть N предметов и M рюкзаков ($M \leqslant N$). 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.*

Формулировка Задачи

Максимизировать $\sum_{i=1}^{M} \sum_{j=1}^{N} p_j x_{ij}$

так, чтобы $\sum_{i=1}^{N} w_j x_{ij} \leqslant W_i$ выполнялось для всех $i = 1, 2, \dots, N$.

$\sum_{j=1}^{M} x_{ij} = 1$ для всех $i = 1, 2, \dots, N$.

$x_{ij} = 1$ если j предмет назначен i рюкзаку. Иначе $x_{ij} = 0$.

Варианты решения

Применение динамического программирования, для задач данного типа нецелесообразно. Используются вариации метода ветвей и границ.

Задача о назначении

Задача:

Задача о назначении *(англ. Generalized Assignment Problem) — Наиболее общая задача семейства. Отличается от мультипликативного рюкзака тем, что каждый предмет имеет различные характеристики в зависимости от рюкзака, куда его помещают. Есть N предметов и M рюкзаков ($M \leqslant N$). У каждого рюкзака своя вместимость W_i, у j предмета p_{ij} стоимость и вес, при помещении его в i рюкзак, равны p_{ij} и w_{ij} соответственно.*

Весьма важная задача, так как она моделирует оптимальное распределение различных задач между вычислительными блоками.

Формулировка Задачи

Максимизировать стоимость выбранных предметов $\sum_{i=1}^{M} \sum_{j=1}^{N} p_{ij} x_{ij}$,

при выполнении условия совместности $\sum_{j=1}^{N} w_{ij} x_{ij} \leqslant W_i \qquad i = 1, \dots, M$.

$\sum_{i=1}^{M} x_{ij} \leqslant 1 \qquad j = 1, \dots, N$.

$x_{ij} \in \{0, 1\} \qquad i = 1, \dots, N, \quad j = 1, \dots, N$

Варианты решения

Применение динамического программирования нецелесообразно. Наиболее используем метод ветвей и границ.

См. также

- Класс NP
- Метод четырех русских для умножения матриц
- Задача о редакционном расстоянии, алгоритм Вагнера-Фишера
- Meet-in-the-middle

Источники информации

- [redacted]
- Silvano Martello, Paolo Toth. Knapsack Problems: Algorithms and Computer Implementations — 1990 г. — ISBN 0-471-92420-2

- ↑ [redacted]
- ↑ Pisinger D (1999). "Linear Time Algorithms for Knapsack Problems with Bounded Weights". *Journal of Algorithms*, Volume 33, Number 1, October 1999, pp. 1–14
- ↑ Koiliaris, Konstantinos; Xu, Chao (2015-07-08). "A Faster Pseudopolynomial Time Algorithm for Subset Sum".
- ↑ Bringmann K. A near-linear pseudopolynomial time algorithm for subset sum[C]//Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2017: 1073-1084

Категории: Дискретная математика и алгоритмы | Динамическое программирование