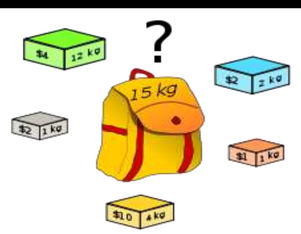
Chggfhfg

2017

PISL 09

PISL09. Тренинг дин. программирования.



PISL09. Тренинг дин. программирования.

- Дискретный рюкзак с повторениями (с практикой A)
- Дискретный рюкзак без повторений (с практикой B)
- Ступеньки (с практикой C)

- Множественное умножение матриц (теоретически)

- Задачи (A) (B) (C)

- **Материалы:** http://tinyurl.com/ei-pisl
- **Github:** https://github.com/Khmelov/PISL2017-01-26

PISL09. Тренинг дин. программирования.

ВикипедиЯ
Свободная энциклопедия

Задача о ранце

Материал из Википедии — свободной энциклопедии

[править | править вики-текст]

Задача о ранце (или **задача о рюкзаке**) — одна из NP-трудных задач комбинаторной оптимизации. Своё название получила от конечной цели: уложить как можно большее число ценных вещей в рюкзак при условии, что вместимость рюкзака ограничена. С различными вариациями задачи о ранце можно столкнуться в экономике, прикладной математике, криптографии и логистике.

В общем виде задачу можно сформулировать так: из заданного множества предметов со свойствами «стоимость» и «вес» требуется отобрать подмножество с максимальной полной стоимостью, соблюдая при этом ограничение на суммарный вес.

Содержание [показать]

Классическая постановка задачи [править | править вики-текст]

Пусть имеется набор предметов, каждый из которых имеет два параметра — вес и ценность. Также имеется рюкзак определенной вместимости. Задача заключается в том, чтобы собрать рюкзак с максимальной ценностью предметов внутри, соблюдая при этом ограничение рюкзака на суммарный вес.

Математически задача формулируется следующим образом: имеется n грузов. Для каждого i-го груза определён его **вес** $w_i > 0$ и **ценность** $v_i > 0$. $i = 1, 2, \ldots, n$. Ограничение суммарного веса предметов в рюкзаке задаётся **грузоподъёмностью** W. Необходимо

максимизировать $\sum_{i=1}^{n} v_i x_i$

с ограничениями $\sum_{i=1}^{n} w_i x_i \leq W$ и $x_i \in \{0, 1\}$ [1]

Пример задачи о ранце: необходимо уложить коробки в ранец вместимостью 15 кг так, чтобы стоимость уложенных коробок была максимальной

Варианты задачи о ранце [править | править вики-текст]

Основная статья: ***Список задач о ранце***

Постановка задачи допускает большое количество обобщений, в зависимости от условий, наложенных на рюкзак, предметы или их выбор. Наиболее популярными разновидностями являются следующие:

1. Рюкзак 0-1 (англ. *0-1 Knapsack Problem*)[2]: не более одного экземпляра каждого предмета. B
2. Ограниченный рюкзак (англ. *Bounded Knapsack Problem*)[3]: не более заданного числа экземпляров каждого предмета.
3. Неограниченный рюкзак (англ. *Unbounded Knapsack Problem*)[3]: произвольное количество экземпляров каждого предмета. A
4. Рюкзак с мультивыбором (англ. *Multiple-choice Knapsack Problem*)[4]: предметы разделены на группы, и из каждой группы требуется выбрать только один предмет.
5. Мультипликативный рюкзак (англ. *Multiple Knapsack Problem*)[5]: есть несколько рюкзаков, каждый со своим максимальным весом. Каждый предмет можно положить в любой рюкзак или оставить.
6. Многомерный рюкзак (англ. *Multy-dimensional knapsack problem*): вместо веса дано несколько разных ресурсов (например, вес, объём и время укладки). Каждый предмет тратит заданное количество каждого ресурса. Надо выбрать подмножество предметов так, чтобы общие затраты каждого ресурса не превышали максимума по этому ресурсу, и при этом общая ценность предметов была максимальна[6].
7. Квадратичная задача о рюкзаке (англ. *Quadratic knapsack problem*): суммарная ценность задается неотрицательно определённой квадратичной формой[7].

PISL09. Тренинг дин. программирования.

Напомним план для дин. прогр.

1. Какие значения мы вычисляем (что ищем)
2. Как их вычислять (какое рекуррентное соотношение)
3. Какие начальные значения (инициализация рекуррентных соотношений)
4. Направление расчета (рекурсия или итерация)
5. Где искать ответ

Подробнее о плане решения задач на динамическое программирование: https://youtu.be/iKj-xl4enLw?t=20m

Chggfhfg

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

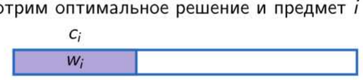
30	16	
6	4	всего: 46 руб.

без повторений

Chggfhfg

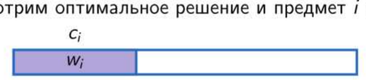
Рюкзак с повторениями: подзадачи

- Рассмотрим оптимальное решение и предмет i в нём:



Рюкзак с повторениями: подзадачи

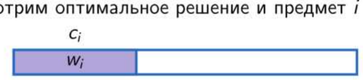
- Рассмотрим оптимальное решение и предмет i в нём:



- 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:

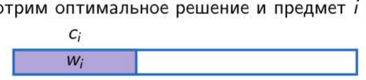


- Если вытащить данный предмет из рюкзака, то мы получим оптимальное заполнение рюкзака вместимости $W - w_i$ («вырезать и вставить»).
- Подзадачи:

$D[w]$ = 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:

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

- Тогда

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

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 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

Chggfhfg

PISL09. Тренинг дин. программирования.

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

PISL09. Тренинг дин. программирования.

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

PISL09. Тренинг дин. программирования.

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

PISL09. Тренинг дин. программирования.

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

Chggfhfg

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 $0 \le w \le W$ и $0 \le i \le n$, $D[w, i]$ — максимальная стоимость рюкзака вместимости w, если разрешено использовать только предметы $1, \ldots, i$.
- Предмет 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]
```

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

Chggfhfg

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.

Chggfhfg

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

```
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 H[w]
```


Время работы

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

Chggfhfg

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 $O(n2^{\log W})$.

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.

Перемножение последовательности матриц

Вход: последовательность n матриц $A_1, \ldots, A_n$, которые нужно перемножить.

Выход: порядок умножения, минимизирующий стоимость умножения.

Chggfhfg

Замечания

- Обозначим размеры матриц $A_1, \ldots, A_n$ через

$$m_0 \times m_1, m_1 \times m_2, \ldots, m_{n-1} \times m_n,$$

соответственно. То есть размер A_i есть $m_{i-1} \times m_i$.

- Умножение матриц не коммутативно (в общем случае, $A \times B \neq B \times A$), но ассоциативно:

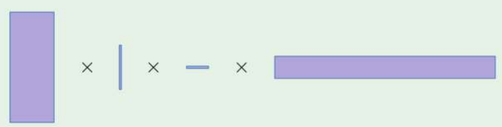
$$A \times (B \times C) = (A \times B) \times C.$$

- Значит, $A \times B \times C \times D$ может быть вычислено как

$$(A \times B) \times (C \times D) \text{ или } (A \times (B \times C)) \times D.$$

- Стоимость умножения двух матриц размеров $p \times q$ и $q \times r$ будем считать pqr.

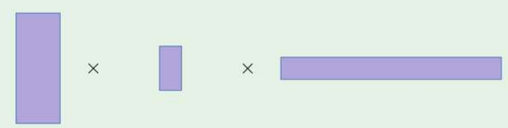
Пример: $A \times ((B \times C) \times D)$



A 50×20 | B 20×1 | C 1×10 | D 10×100

стоимость:

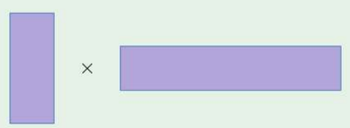
Пример: $A \times ((B \times C) \times D)$



A 50×20 | $B \times C$ 20×10 | D 10×100

стоимость: $20 \cdot 1 \cdot 10$

Пример: $A \times ((B \times C) \times D)$



A 50×20 | $B \times C \times D$ 20×100

стоимость: $20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100$

PISL09. Тренинг дин. программирования.

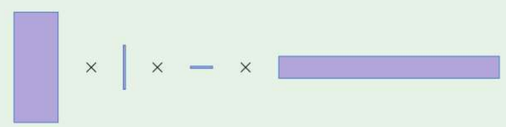
Пример: $A \times ((B \times C) \times D)$

$A \times B \times C \times D$
50×100

стоимость: $20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100 = 120\,200$

PISL09. Тренинг дин. программирования.

Пример: $(A \times B) \times (C \times D)$



A 50×20 | B 20×1 | C 1×10 | D 10×100

стоимость:

PISL09. Тренинг дин. программирования.

Пример: $(A \times B) \times (C \times D)$

$A \times B$ 50×1 | C 1×10 | D 10×100

стоимость: $50 \cdot 20 \cdot 1$

PISL09. Тренинг дин. программирования.

Пример: $(A \times B) \times (C \times D)$

$A \times B$ 50×1 | $C \times D$ 1×100

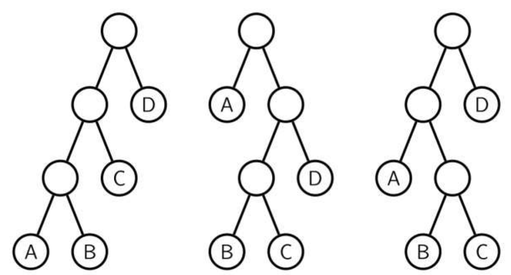
стоимость: $50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100$

Пример: $(A \times B) \times (C \times D)$

$A \times B \times C \times D$
50×100

стоимость: $50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100 = 7\,000$

Порядки как строго двоичные деревья



$((A \times B) \times C) \times D$ $A \times ((B \times C) \times D)$ $(A \times (B \times C)) \times D$

Подзадачи и рекуррентное соотношение

- Для $1 \le i \le j \le n$, пусть

  $D[i,j] =$ мин. стоимость вычисления $A_i \times A_{i+1} \times \ldots \times A_j$.

- Корень поддерева разбивает его на два поддерева: $A_i \times \ldots \times A_k$ и $A_{k+1} \times \ldots \times A_j$ (для некоторого $i \le k < j$).
- Рекуррентное соотношение:

$$D[i,j] = \min_{i \le k < j} \{D[i,k] + D[k+1,j] + m_{i-1} \cdot m_k \cdot m_j\}.$$

Дин. прог. сверху вниз

Инициализация

создать таблицу $D[1 \ldots n, 1 \ldots n] \leftarrow [\infty, \ldots, \infty]$

Дин. прог. сверху вниз

Инициализация

создать таблицу $D[1\ldots n, 1\ldots n] \leftarrow [\infty, \ldots, \infty]$

Функция MATRIXMULTTD(i, j)

```
если D[i,j] = ∞:
  если i = j:     D[i,j] ← 0
  иначе:
    для k от i до j − 1:
      ℓ ← MATRIXMULTTD(i, k)
      r ← MATRIXMULTTD(k + 1, j)
      D[i,j] ← min(D[i,j], ℓ + r + m_{i−1} m_k m_j)
вернуть D[i,j]
```


Дин. прог. сверху вниз

Инициализация

создать таблицу $D[1\ldots n, 1\ldots n] \leftarrow [\infty, \ldots, \infty]$

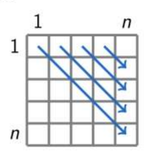
Функция MATRIXMULTTD(i, j)

```
если D[i,j] = ∞:
  если i = j:     D[i,j] ← 0
  иначе:
    для k от i до j − 1:
      ℓ ← MATRIXMULTTD(i, k)
      r ← MATRIXMULTTD(k + 1, j)
      D[i,j] ← min(D[i,j], ℓ + r + m_{i−1} m_k m_j)
вернуть D[i,j]
```

Время работы: $O(n^3)$.

Порядок подзадач

- Хотим идти от меньших подзадач к бóльшим.
- Размером подзадачи естественно считать требующееся количество умножений: $j - i$.
- Возможный порядок:



Дин. прог. снизу вверх

Функция MATRIXMULTBU$(m_0, m_1, \ldots, m_n)$

```
создать массив D[1...n, 1...n] ← [∞, ..., ∞]
для i от 1 до n:
  D[i,i] ← 0
для s от 1 до n − 1:
  для i от 1 до n − s:
    j ← i + s
    для k от i до j − 1:
      D[i,j] ← min(D[i,j], D[i,k] + D[k+1,j] + m_{i−1} m_k m_j)
вернуть D[1,n]
```

Chggfhfg

Дин. прог. снизу вверх

Функция MATRIXMULTBU($m_0, m_1, \ldots, m_n$)

```
создать массив D[1...n, 1...n] ← [∞, ..., ∞]
для i от 1 до n:
  D[i, i] ← 0
для s от 1 до n − 1:
  для i от 1 до n − s:
    j ← i + s
    для k от i до j − 1:
      D[i, j] ← min(D[i, j], D[i, k] + D[k + 1, j] + m_{i−1} m_k m_j)
вернуть D[1, n]
```

Время работы: $O(n^3)$.

Пример

$m_0 = 50,\ m_1 = 20,\ m_2 = 1,\ m_3 = 10,\ m_4 = 100$

	1	2	3	4
1	0	1000	1500	7000
2		0	200	3000
3			0	1000
4				0

Задание А.

```
/*
Задача на программирование: рюкзак с повторами

Первая строка входа содержит целые числа
    1<=W<=100000     вместимость рюкзака
    1<=n<=300        сколько есть вариантов золотых слитков
                     (каждый можно использовать множество раз).
Следующая строка содержит n целых чисел, задающих веса слитков:
  0<=w[1]<=100000 ,..., 0<=w[n]<=100000

Найдите методами динамического программирования
максимальный вес золота, который можно унести в рюкзаке.


Sample Input:
10 3
1 4 8
Sample Output:
10

Sample Input 2:
15 3
2 8 16
Sample Output 2:
14
```


Задание Б.

```
/*
Задача на программирование: рюкзак без повторов

Первая строка входа содержит целые числа
    1<=W<=100000     вместимость рюкзака
    1<=n<=300        число золотых слитков
                     (каждый можно использовать только один раз).
Следующая строка содержит n целых чисел, задающих веса каждого из слитков:
  0<=w[1]<=100000 ,..., 0<=w[n]<=100000

Найдите методами динамического программирования
максимальный вес золота, который можно унести в рюкзаке.

Sample Input:
10 3
1 4 8
Sample Output:
9

*/
```

Chggfhfg

Задание C.

```
/*
Даны число 1<=n<=100 ступенек лестницы и
целые числа -10000<=a[1],…,a[n]<=10000, которыми помечены ступеньки.
Найдите максимальную сумму, которую можно получить, идя по лестнице
снизу вверх (от нулевой до n-й ступеньки), каждый раз поднимаясь на
одну или на две ступеньки.

Sample Input 1:
2
1 2
Sample Output 1:
3

Sample Input 2:
2
2 -1
Sample Output 2:
1

Sample Input 3:
3
-1 2 1
Sample Output 3:
3

*/
```