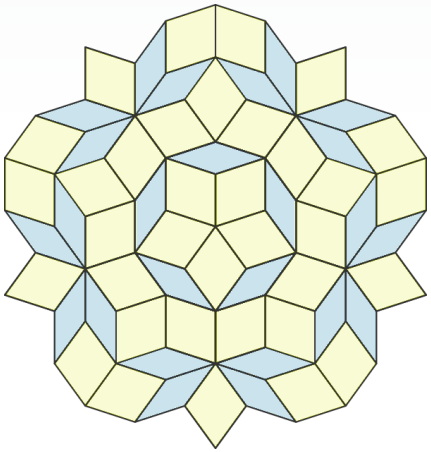
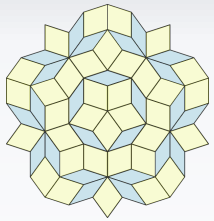


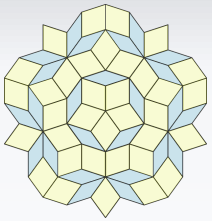
Сопоставление с образцом

Летняя практика, Переславль–Залесский
4–8 июля 2022 г.



Немного терминологии из комбинаторики слов

- Строка w в алфавите Σ ($w \in \Sigma^*$) — слово. ε обозначает пустое слово.
- Подслово v слова $w_1 v w_2$ — делитель.
- Если $w = \underbrace{v v \ldots v}_{n}$ — тогда говорим, что w — n-ая степень v. Пишем $w = v^n$. Например, квадрат — это слово, представимое в виде vv (в стандартном допущении о том, что $v \neq \varepsilon$).
- Простое слово — не являющееся степенью никакого слова, кроме себя самого.
- Длина слова w обозначается $|w|$. Количество вхождений в w буквы t — как $|w|_t$.

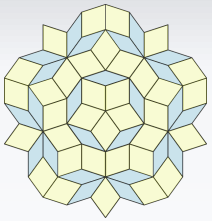


Switch – case

`switch` в Python 3.10:

```
match len(answer):
    case 0:
        print('решений нет')
    case 1:
        # что-то сделать с решением
    case _:
        print('решений больше одного')
```

Пока что ничего особенного, просто аналог `if - elif - else` цепочек.

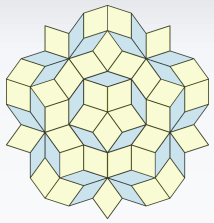


Сопоставление с образцом

Но можно и так:

```
match answer:
    case []:
        print('решений нет')
    case [x]:
        # что-то сделать с x
    case [x, *_]:
        print('решений больше одного')
```

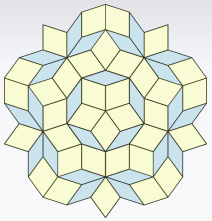
Появляется возможность сравнивать `answer` не с константой, а с параметризованным конструктором (в данном случае списком), причём с возможностью сразу же заглянуть внутрь списка (достать первый элемент).



Декларативные объявления в Haskell

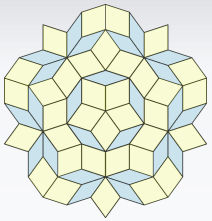
```
f [] = ...
f [x] = ...
f (x : _) = ...
```

- Частичный разбор терма начиная с внешнего конструктора.
- Сопоставление сверху вниз (аналогично цепочке `if–elif–else`).
- Не перестановочные предложения: третье в том числе описывает и случай, когда список одноэлементный (т.к. алгебраически список определяется конструкторами `:` (т.е. `cons`) и `[]`, т.е. `nil`).



Стандартные ограничения

- Отсутствие повторных переменных в образцах (линейность):
  - `f x x` — нельзя.
  - В языке Prolog — можно.
- Однозначный разбор по внешнему конструктору (так называемое сопоставление с образцом в свободной алгебре):
  - `f x ++ ['A'] ++ y` — нельзя. Здесь `++` — операция конкатенации списков / строк.
  - views Wadler'a — в дальнейшем язык Egison — можно.



Нелинейность и несвобода

- Проблемы с перестановочностью (даже неявной):

```
f x y x = ...
f x x y = ...
f y x x = ...
```

- Проблемы с однозначностью ответа:

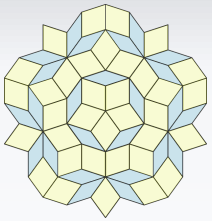
```
f x ++ ['A'] ++ y = ...
...
f ['A','A','A',...,'A']
```

(в языке PROLOG возвращаются все результаты)

- Неочевидный алгоритм разбора по образцу:

```
f y0 ++ ('A' : x) ++ y1 ++ ('A' : x) ++ y2
```

Имея изначальным аргументом `f` единственный список (строку литералов), мы вынуждены искать представление этого списка как конкатенации пяти списков.



Регулярные выражения

Проблема неоднозначности разбора строковых данных уже решалась в рамках построения механизма сопоставления с регулярными выражениями.

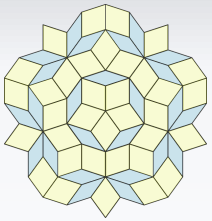
- Жадные квантификаторы — забирают максимально длинную подстроку

```
match = re.search(r'\( .* \)', r'0( (12)3)4(56 )7')
# match[0] = ((12)3)4(56)
```

- Ленивые квантификаторы — забирают максимально короткую подстроку

```
match = re.search(r'\( .*? \)', r'0( (12 )3)4(56)7')
# match[0] = ((12)
```

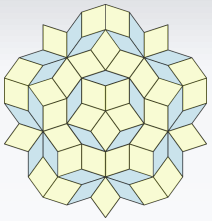
Здесь `.` — произвольная буква, `*` — квантификатор: повторять образец 0 или больше раз, `\(` и `\)` — экранированные представления круглых скобок.



Ассоциативные образцы

Для краткости ++ опускается. Например: `x'A'y` — сокращённая запись для `x ++ ['A'] ++ y`.

- Сопоставление ленивое, сверху вниз.
- Эффективный доступ к строке с начала, с конца или с середины (например, при использовании суффиксного массива). Условный "шаг" — вызов сопоставления. С точки зрения реализации, "шаг" имеет сложность, большую, чем $O(1)$.
- Строки — очевидно. Как добавить леса с доступом "с середины"?

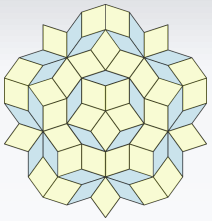


Ассоциативные образцы

- Сопоставление ленивое, сверху вниз.
- Эффективный доступ к строке с начала, с конца или с середины (например, при использовании суффиксного массива). Условный "шаг" — вызов сопоставления. С точки зрения реализации, "шаг" имеет сложность, большую, чем $O(1)$.
- Строки — очевидно. Как добавить леса с доступом "с середины"? Решение: дополнительный конструктор, добавляющий глубину вложенности.

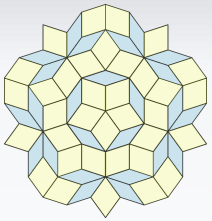
Дан список списков. Выделить список, содержащий букву `'A'`.

`(list x1 (list z1 'A' z2) x2)` — с тегами (АТД);
`(x1 (z1 'A' z2) x2)` — без тегов (ака Рефал);



Примеры образцов

- Ленивый образец, находящий две одинаковые закавыченные последовательности?
- Образец, проверяющий, есть ли в слове квадрат (т.е. дважды повторяющееся подслово)?



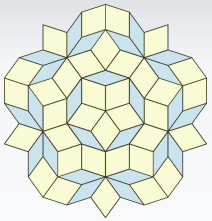
Примеры образцов

- Ленивый образец, находящий две одинаковые закавыченные последовательности?

  `x1 " x2 " x3 " x2 " x4`

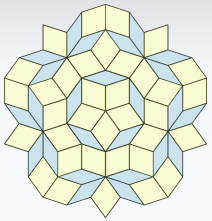
  Заметим, что если внутри значения `x2` окажутся хотя бы одни кавычки, то существует более «ленивое» сопоставление (с той же длиной значения `x1`, но меньшей — `x2`).

- Образец, проверяющий, есть ли в слове квадрат (т.е. дважды повторяющееся подслово)?



Примеры образцов

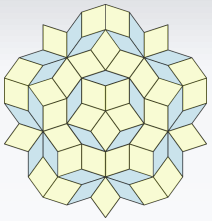
- Ленивый образец, находящий две одинаковые закавыченные последовательности?
  `x1 " x2 " x3 " x2 " x4`
- Образец, проверяющий, есть ли в слове квадрат (т.е. дважды повторяющееся подслово)?
  Первая проба: `x1 x2 x2 x3`



Примеры образцов

- Ленивый образец, находящий две одинаковые закавыченные последовательности?
  `x1` `" x2 "` `x3` `" x2 "` `x4`

- Образец, проверяющий, есть ли в слове квадрат (т.е. дважды повторяющееся подслово)?
  Первая проба: `x1` `x2` `x2` `x3`
  Fail! Тривиально, из-за наличия в моноиде единицы — пустого слова. Требуется подчеркнуть непустоту — сказать, что в квадрат входит хотя бы один символ.
  Разделим переменные образца на два класса:
  - `e.name` — expression, сопоставляется с чем угодно;
  - `t.name` — term, сопоставляется только с символом либо скобочным выражением.



Примеры образцов

- Ленивый образец, находящий две одинаковые закавыченные последовательности?

  `x1` `" x2 "` `x3` `" x2 "` `x4`

- Образец, проверяющий, есть ли в слове квадрат (т.е. дважды повторяющееся подслово)?

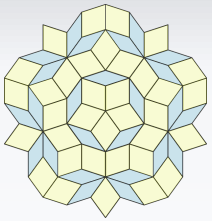
  Разделим переменные образца на два класса:

  - `e.name` — expression, сопоставляется с чем угодно;
  - `t.name` — term, сопоставляется только с символом либо скобочным выражением.

  Тогда решение задачи выглядит так:

  `e.x1` `t.y e.x2` `t.y e.x2` `e.x3`

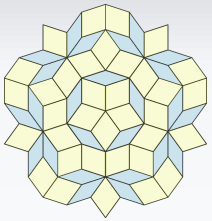
  Далее сокращаем `e.name` просто до `name` (начинается не с `t`), а `t.name` — до `tname`.



Функции над образцами

- Функция, находящая в аргументе две одинаковые закавыченные последовательности, и возвращающая одну из них?
- Функция, находящая в слове квадрат (т.е. дважды повторяющееся подслово)?

Далее сокращаем `e.name` просто до `name` (начинается не с `t`), а `t.name` — до `tname`.



Функции над образцами

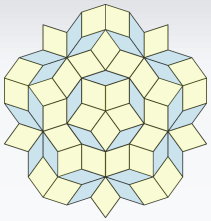
- Функция, находящая в аргументе две одинаковые закавыченные последовательности, и возвращающая одну из них?

```
F {x1 " x2 " x3 " x2 " x4 = x2;}
```

- Функция, находящая в слове квадрат (т.е. дважды повторяющееся подслово)?

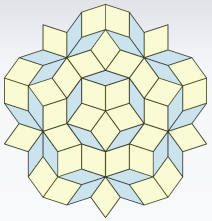
```
F {e.x1 t.y e.x2 t.y e.x2 e.x3 = t.y e.x2;}
```

Далее сокращаем `e.name` просто до `name` (начинается не с `t`), а `t.name` — до `tname`.



Задача Корлюкова ++

Дано двоичное число длины 3. Записать функцию, возвращающую в двоичной форме (с лидирующими нулями) число единиц в нём.

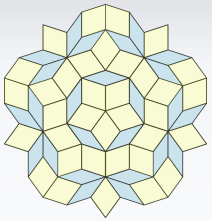


Задача Корлюкова ++

Дано двоичное число длины 3. Записать функцию, возвращающую в двоичной форме (с лидирующими нулями) число единиц в нём.

Решение «в лоб»:

```
F { 0 0 0 = 0 0;
    0 0 1 = 0 1;
    0 1 0 = 0 1;
    0 1 1 = 1 0;
    1 0 0 = 0 1;
    1 0 1 = 1 0;
    1 1 0 = 1 0;
    1 1 1 = 1 1; }
```



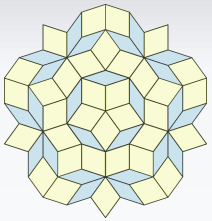
Задача Корлюкова ++

Дано двоичное число длины 3. Записать функцию, возвращающую в двоичной форме (с лидирующими нулями) число единиц в нём.

Решение «в лоб»:

```
F { 0 0 0 = 0 0;
    0 0 1 = 0 1;
    0 1 0 = 0 1;
    0 1 1 = 1 0;
    1 0 0 = 0 1;
    1 0 1 = 1 0;
    1 1 0 = 1 0;
    1 1 1 = 1 1; }
```

- Два нуля влекут результат `0 1`;
- Две единицы влекут результат `1 0`;
- Осталось разобраться с `t t t`. Но такие данные порождают всегда `t t`.

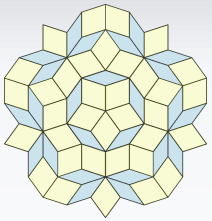


Задача Корлюкова ++

Дано двоичное число длины 3. Записать функцию, возвращающую в двоичной форме (с лидирующими нулями) число единиц в нём.

Второе приближение:

```
F { t t t = t t;
    x1 0 x2 0 x3 = 0 1;
    x1 1 x2 1 x3 = 1 0;
  }
```



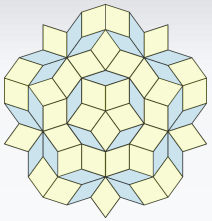
Задача Корлюкова ++

Дано двоичное число длины 3. Записать функцию, возвращающую в двоичной форме (с лидирующими нулями) число единиц в нём.

Второе приближение:

```
F { t t t = t t;
    x1 0 x2 0 x3 = 0 1;
    x1 1 x2 1 x3 = 1 0;
  }
```

- Видно, что вторая и третья строчки почти одинаковы — на первом месте в результате стоит выделенное значение, на втором — другое;
- Длина строки `x1 x2 x3` всегда равна 1, и это именно то другое значение!

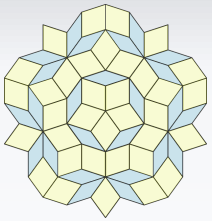


Задача Корлюкова ++

Дано двоичное число длины 3. Записать функцию, возвращающую в двоичной форме (с лидирующими нулями) число единиц в нём.

Почти решение:

```
F { t t t = t t;
    x1 t x2 t x3
      = t x1 x2 x3;
  }
```



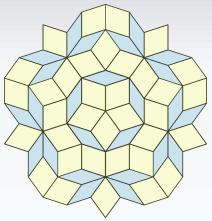
Задача Корлюкова ++

Дано двоичное число длины 3. Записать функцию, возвращающую в двоичной форме (с лидирующими нулями) число единиц в нём.

Почти решение:

```
F { t t t = t t;
    x1 t x2 t x3
      = t x1 x2 x3;
  }
```

- А теперь видно, что и первая строчка не нужна — в ней делается то же, что и во второй.
- Итоговая функция содержит всего одну строчку:
  `x1 t x2 t x3 = t x1 x2 x3`.
  Причём `xi` справа от знака = можно располагать как угодно относительно друг друга, ответ всё равно будет правильным.

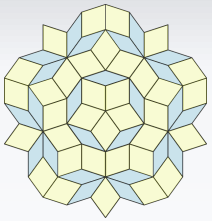


Задача Корлюкова ++

Чуть-чуть усложним задачу.

Дано двоичное число длины 4. Записать в двоичной форме (с лидирующими нулями) число единиц в нём.

Видно, что за исключением случая четырёх единиц, сгодилось бы предыдущее решение, если бы мы умели выкидывать из образца один ноль и собирать всё остальное в новый образец.



Задача Корлюкова ++

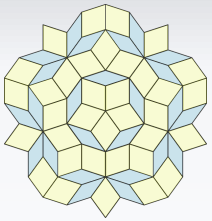
Чуть-чуть усложним задачу.

Дано двоичное число длины 4. Записать в двоичной форме (с лидирующими нулями) число единиц в нём.

Видно, что за исключением случая четырёх единиц, сгодилось бы предыдущее решение, если бы мы умели выкидывать из образца один ноль и собирать всё остальное в новый образец.

Как описать образцы для элементов образцов?

[Образец](, [Выражение] : [Образец])*



Задача Корлюкова ++

Чуть-чуть усложним задачу.

Дано двоичное число длины 4. Записать в двоичной форме (с лидирующими нулями) число единиц в нём.

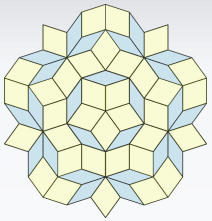
Видно, что за исключением случая четырёх единиц, сгодилось бы предыдущее решение, если бы мы умели выкидывать из образца один ноль и собирать всё остальное в новый образец.

Как описать образцы для элементов образцов?

[Образец](, [Выражение] : [Образец])*

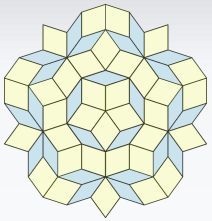
Решение задачи Корлюкова++ в этих терминах:

```
F {1 1 1 1 = 1 0 0;
   x1 0 x2
   , x1 x2 : z1 t z2 t z3 = 0 t z1 z2 z3; }
```



Другие образцы

- Слово содержит как букву 'A', так и букву 'B'.
- Два слова являются циклическими перестановками (т.е. $w_1 = uv$, $w_2 = vu$).
- Слово не содержит ничего, кроме (произвольного числа) букв 'A'.
- Два слова являются степенями одного и того же слова ($w_1 = v^i$, $w_2 = v^j$).



Другие образцы

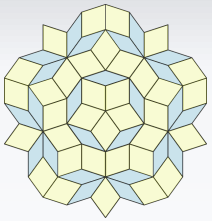
- Слово содержит как букву 'A', так и букву 'B'.

  Ответ: образец `x1 'A' x2, x1 x2: z1 'B' z2`
  (поскольку 'A' точно не содержит ничего от 'B')

- Два слова являются циклическими перестановками (т.е. $w_1 = uv$, $w_2 = vu$).

- Слово не содержит ничего, кроме (произвольного числа) букв 'A'.

- Два слова являются степенями одного и того же слова ($w_1 = v^i$, $w_2 = v^j$).



Другие образцы

- Слово содержит как букву 'A', так и букву 'B'.

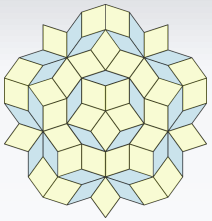
  Ответ: образец `x1 'A' x2, x1 x2: z1 'B' z2`
  (поскольку 'A' точно не содержит ничего от 'B')

- Два слова являются циклическими перестановками (т.е. $w_1 = uv$, $w_2 = vu$).

  Тут решение тривиальное: `(x1 x2) (x2 x1)`.

- Слово не содержит ничего, кроме (произвольного числа) букв 'A'.

- Два слова являются степенями одного и того же слова ($w_1 = v^i$, $w_2 = v^j$).



Другие образцы

- Слово содержит как букву 'A', так и букву 'B'.

  Ответ: образец `x1 'A' x2, x1 x2: z1 'B' z2`
  (поскольку 'A' точно не содержит ничего от 'B')

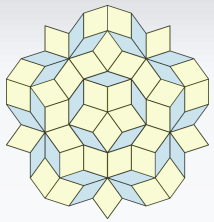
- Два слова являются циклическими перестановками (т.е. $w_1 = uv$, $w_2 = vu$).

  Тут решение тривиальное: `(x1 x2) (x2 x1)`.

- Слово не содержит ничего, кроме (произвольного числа) букв 'A'.

  Решение нетривиальное: `x, 'A' x : x 'A'`.

- Два слова являются степенями одного и того же слова ($w_1 = v^i$, $w_2 = v^j$).



Другие образцы

- Слово содержит как букву 'A', так и букву 'B'.

  Ответ: образец `x1 'A' x2, x1 x2: z1 'B' z2`
  (поскольку 'A' точно не содержит ничего от 'B')

- Два слова являются циклическими перестановками (т.е. $w_1 = uv$, $w_2 = vu$).

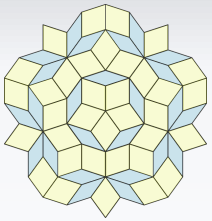
  Тут решение тривиальное: `(x1 x2) (x2 x1)`.

- Слово не содержит ничего, кроме (произвольного числа) букв 'A'.

  Решение нетривиальное: `x, 'A' x : x 'A'`.

- Два слова являются степенями одного и того же слова ($w_1 = v^i$, $w_2 = v^j$).

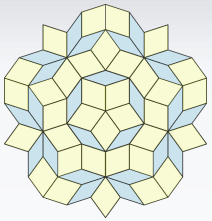
  По аналогии с предыдущим: `(x1) (x2), x1 x2 : x2 x1`.



Ещё несколько задач

Какие языки описываются следующими образцами?

- `x1, x1 A B : B A x1, x1 : x2 x2`
- `x1, x1 x1: t1 x2 x1 x3, x1: t1 x2 t2 x4`



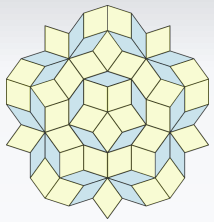
Ещё несколько задач

Какие языки описываются следующими образцами?

- `x1, x1 A B : B A x1, x1 : x2 x2`

  Пустой язык. Поскольку уравнение (а это именно уравнение) `x1 A B = B A x1` имеет решения вида `B (A B)*`, а они все — нечётной длины.

- `x1, x1 x1: t1 x2 x1 x3, x1: t1 x2 t2 x4`



Ещё несколько задач

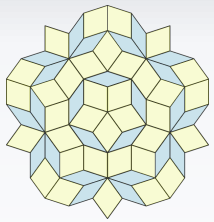
Какие языки описываются следующими образцами?

- `x1, x1 A B : B A x1, x1 : x2 x2`

  Пустой язык. Поскольку уравнение (а это именно уравнение) `x1 A B = B A x1` имеет решения вида `B (A B)*`, а они все — нечётной длины.

- `x1, x1 x1: t1 x2 x1 x3, x1: t1 x2 t2 x4`

  Язык слов вида w^s, где $s \geqslant 2$. Почему они подходят, понять легко, а вот почему не подходят другие слова, проще понять, изучив основы комбинаторики слов.



Проектирование структур с образцами

- Вопрос достижимости образца:

```
f (A : x) = t1
f [] = t2
f [A] = t3
```

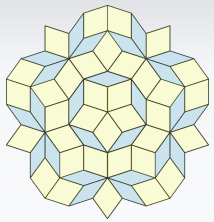
- Вопрос накрытия образцами:

```
f ((x : y) : z) = t1
f [] = t2
```

- Вопрос перестановочности образцов:

```
f (x : (A : y)) = t1
f (A : (y : z)) = t2
```

...а с отказом от свободы и единственности вхождений переменных эти вопросы становятся намного сложнее.



Двумерный случай

- Определим отношение соседства как отношение примыкания в матрице.
- С помощью образцов зададим правила переписывания.

Применение двумерных систем переписывания по образцу:

- Эволюционные алгоритмы (например, игра «жизнь» Конвея) (классические клеточные автоматы).
- Генерация случайных объектов (стохастические клеточные автоматы).