期中讲评

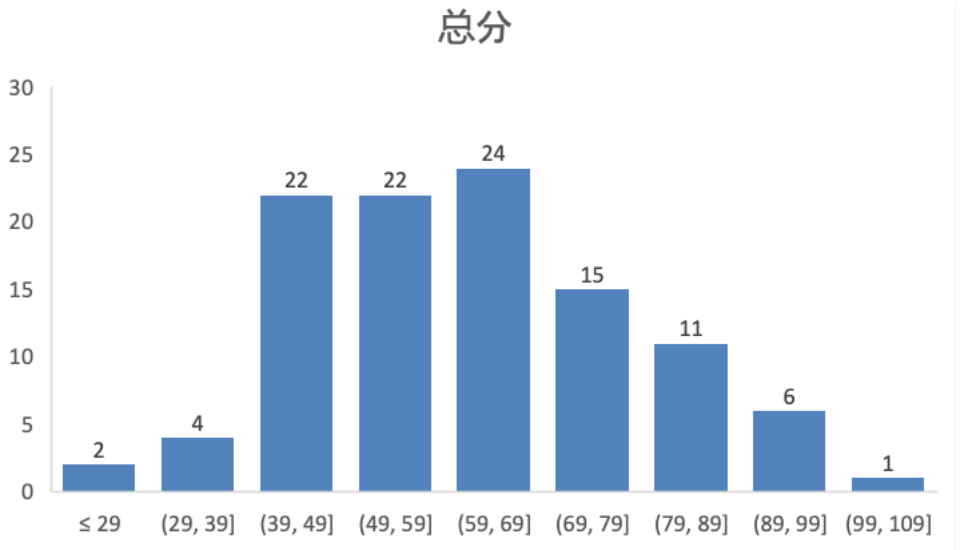
陈钦霖、陈智骐、林哲浩、蒋圣翊、黄奕诚
2020.11.26

考情简况

平均分——61.6分

中位分——61分

最高分——100分



题目	P1	P2	P3	P4	P5	P6	P7	附加
每题总分	18	12	12	5	5	27	21	3
均分	12.03	5.75	3.40	4.44	4.87	17.45	11.36	2.10
得分率	66.82%	47.94%	28.35%	88.79%	97.38%	64.62%	54.12%	70.09%

考生认为的难度 

助教认为的难度 

Problem 1

—— What Would Python Display

```
2**11 - 28
```

正确答案：2020

- $2^{11} - 28 = 1024 \times 2 - 28 = 2048 - 28 = 2020$

错误答案1：996

- `2**10 - 28` ?

错误答案2：-6

- `2*11 - 28` ?

`True and not 1/0`

考点：短路求值

- Lab01 – Question2
- Slides – 03-Control

Question 2: Veritasiness

```
>>> True and 13
______

>>> False or 0
______

>>> not 10
______

>>> not None
______

>>> True and 1 / 0 and False
______

>>> True or 1 / 0 or False
______
```

Boolean Expressions

Boolean expressions contain special operators `and`, `or`, `not`

- `<exp1> and <exp2> and <exp3> and ...`
  - Evaluate to the first false value.
  - If none are false, evaluates to the last expression
- `<exp1> or <exp2> or <exp3> or ...`
  - Evaluate to first true value.
  - If none are true, evaluates to the last expression
- `not <exp>`
  - Evaluates to True if <exp> if a *false value* and False if <exp> is a *true value*

```
-1+1 or [] or 2 or 1/0
```

```
0 or [] or 2 or 1/0
```

```
[] or 2 or 1/0
```

```
2 or 1/0
```

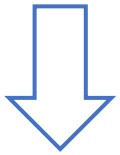
```
2
```

Falsey Value

- `0, None, '', []`

```
True and not 1/0
```

```
not 1/0
```



```
Error
```

正确答案：Error

```
foo('SICP', print)(2020)
```

考点1：简单的内部函数

考点2：短路求值

考点3：print/return

- Lab01 – 2.1.2 return and print

```
def foo(x, f):
    def bar(g):
        return f(x) or g
    return bar
```

```
foo('SICP', print)(2020)
```

```
print('SICP') or 2020
```

print函数副作用输出：SICP

```
None or 2020
```

2020

解释器打印：2020

正确答案：

SICP

2020

print('SICP')输出SICP而不是'SICP'

- Lab01 – 2.1.2 return and print

However, unlike a `return` statement, when Python evaluates a `print` expression, the function doe

```
def what_prints():
    print('Hello World!')
    return 'Exiting this function.'
    print('61A is awesome!')

>>> what_prints()
Hello World!
'Exiting this function.'
```

> Notice also that print will display text without the quotes, but return will preserve the quotes.

为什么?

print(obj)的简单原理

```
def print(obj):
    if type(obj) != str:
        obj = str(obj)
    把obj中每个字符输出到屏幕上
```

```
>>> print('SICP')
SICP
```

- 'SICP'中的引号不是字符串中的字符。
- 引号被用来表示字符串字面量。

Python解释器打印表达式的方法

```
>>> 'SICP'
'SICP'
```

```
>>> <expr>
```
转化为
```
print(repr(<expr>))
```

repr(obj)：获取能构造obj的字符串。

```
>>> repr('SICP') == "'SICP'"
True

>>> print(repr('SICP'))
'SICP'
```

自定义的class怎么实现str和repr?

```
class Dog:
    def __init__(self, name):
        self.name = name

>>> d = Dog('Eddie')
>>> print(d) # 等价于 print(str(d))
<__main__.Dog object at ...>
>>> d          # 等价于 print(repr(d))
<__main__.Dog object at ...>
```

自定义的class怎么实现str和repr?

```
class Dog:
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return 'Dog:' + self.name
    def __repr__(self):
        return 'Dog({})'.format(repr(self.name))
```

```
>>> d = Dog('Eddie')
>>> print(d) # 等价于 print(str(d))
Dog: Eddie
>>> d          # 等价于 print(repr(d))
Dog('Eddie')
```

```
min, max = max, min(2, 0)
print(min)
print(max)
```

考点：name绑定，同时赋值，打印函数值

正确答案：

Function
0

错误答案：

None? max? Error?
0

```
t = (0, [1])
t[0] = 2
s = t[1]
t[1].append(2)
s is t[1]
t[1][1:] + [t[0]] + [2, 0]
```

考点：tuple，list

- Slides: 11-Mutable_Values_full
- Lab04

- Slides: 11-Mutable_Values_full

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1,
>>> ooze()
>>> turtle
(1, 2, 3)
```

Nex
chan

```
t = (0, [1])
t[0] = 2
t[1].append(2)
```

[1, 2, 3]

uld be inside!']

The value of an expressi ames or objects

Name change:

```
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it *contains* a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```

```
>>> s = ([1, 2], 3)
>>> s[0][0] = 4
>>> s
```

12

- lab04: 2.1 Mutability

This is important in our discussion of mutability because when we mutate an object, we simply change its state, not its identity.

```
>>> lst1 = [1, 2, 3, 4]
>>> lst2 = lst1
>>> lst1.append(5)
>>> lst2
[1, 2, 3, 4, 5]
>>> lst1 is lst2
True
```

```
s = t[1]
t[1].append(2)
s is t[1]
```

```
t = (0, [1])
t[0] = 2
s = t[1]
t[1].append(2)
s is t[1]
t[1][1:] + [t[0]] + [2, 0]
```

错误答案2

None
Error
None
None
True
[2, 0, 2, 0]

正确答案

Error
True
[2, 0, 2, 0]

错误答案1

False
[2, 2, 2, 0]

```
my_all(True, False)
my_all([])
my_all([True, False])
```

```
def my_all(it):
    for x in it:
        if not x:
            return False
    return True
```

考点：控制流

正确答案：

Error

True

False

错误答案：

Error

False?

True?

以上是我以为的送分题 

```
p = gen_p()
for _ in range(3):
    print(next(p))
p is gen_p()
```

正确答案:

2
3
5
False

考点：控制流，generator

```
def gen_p():
  n = 2
  while True:
    if my_all([n % i != 0
               for i in range(2, n)]):
      yield n
    n += 1
```

- lab05: 2.3 Generators

Separate calls to `countdown` will create distinct generator objects with their own state. Usually, gener
the sequence, create another generator object by calling the generator function again.

```
>>> c1, c2 = countdown(5), countdown(5)
>>> c1 is c2
False
>>> next(c1)
Beginning countdown!
5
>>> next(c2)
Beginning countdown!
5
```

```
p = gen_p()
p is gen_p()
```

这也是送分题 

```
com(3)(m10)(a2)(m10)(20)
```

正确答案：
2020

考点：高阶函数，lambda函数

三种做法：

1. 计算（也不难）
2. 看懂含义（牛!）
3. 猜含义（一线生机）

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
com(3)(m10)(a2)(m10)(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
com(3)(m10)(a2)(m10)(20)
```

```
(λ f: λ g: com(2)(λ x: f(g(x))))(m10)(a2)(m10)(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))


(λ f: λ g: com(2)(λ x: f(g(x))))(m10)(a2)(m10)(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ f: λ g: com(2)(λ x: f(g(x))))(m10)(a2)(m10)(20)
```

```
    (λ g: com(2)(λ x: m10(g(x))))(a2)(m10)(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ g: com(2)(λ x: m10(g(x))))(a2)(m10)(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ g: com(2)(λ x: m10(g(x))))(a2)(m10)(20)
```

```
com(2)(λ x: m10(a2(x))))(m10)(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
com(2)(λ x: m10(a2(x)))(m10)(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
com(2)(λ x: m10(a2(x)))(m10)(20)
```

```
(λ f: λ g: com(1)(λ x: f(g(x)))) \
  (λ x: m10(a2(x)))(m10)(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ f: λ g: com(1)(λ x: f(g(x)))) \
  (λ x: m10(a2(x)))(m10)(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ f: λ g: com(1)(λ x: f(g(x)))) \
  (λ x: m10(a2(x)))(m10)(20)
```

```
com(1)(λ x: (λ x: m10(a2(x)))(m10(x)))(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
com(1)(λ x: (λ x: m10(a2(x)))(m10(x)))(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
com(1)(λ x: (λ x: m10(a2(x)))(m10(x)))(20)
```

```
(λ f: λ x: f(x)) \
  (λ x: (λ x: m10(a2(x)))(m10(x)))(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ f: λ x: f(x)) \
  (λ x: (λ x: m10(a2(x)))(m10(x)))(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ f: λ x: f(x)) \
  (λ x: (λ x: m10(a2(x)))(m10(x)))(20)
```

```
(λ x: (λ x: m10(a2(x)))(m10(x)))(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ x: (λ x: m10(a2(x)))(m10(x)))(20)
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ x: (λ x: m10(a2(x)))(m10(x)))(20)
```

```
(λ x: m10(a2(x)))(m10(20))
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ x: m10(a2(x)))(m10(20))
```

计算!

```
def com(n):
  if n == 1:
    return lambda f: lambda x: f(x)
  return lambda f: lambda g: com(n - 1)(lambda x: f(g(x)))
```

```
(λ x: m10(a2(x)))(m10(20))
```

```
m10(a2(m10(20)))
```

计算!

```
a2 = lambda x: 2 + x
m10 = lambda x: 10 * x
```

```
m10(a2(m10(20)))
```

```
m10(a2((λ x: 10 * x)(20)))
```

```
m10(a2(10 * 20))
```

```
m10(a2(200))
```

```
2020
```

第一题平均失分：-6分

难乎哉？不难也 

Problem 2&3

—— Environment Diagram & Boxing Day

Python Tutor 演示

Problem 4

—— Reverse Digits

倒转一个数位

(5 points) Reverse Digits

Define a function `reverse_digit` that takes an input integer `n` as argument and returns another integer with an reversed order of the digits of `n`.

```
def reverse_digit(n):
    """
    >>> reverse_digit(1240)
    421
    """
    result = _____

    while _____:

        result = _____

        n = _____

    return result
```

参考答案

```
def reverse_digit(n):
    result = 0
    while n > 0:
        result = result * 10 + (n % 10)
        n = n // 10
return result
```

```
def reverse_digit(n):
    result = 0
    while n > 0:
        result = result * 10 + (n % 10)
        n = n / 10 (或者 (n - n % 10) / 10)
return result
```

```
def reverse_digit(n):
    result = 0
    while n > 0:
        result = result * 10 + (n % 10) * pow(len(n), 10)
        n = n // 10
return result
```

```
def reverse_digit(n):
    result = n % 10
    while n >= 10:
        result = result * 10 + (n // 10) % 10
        n = n // 10
return result
```

Problem 5

—— Simple Math

用加法实现乘法!

```
m * n = n + (m - 1) * n
```

e.g.

```
  mult(3, 2)
= plus(2, mult(2, 2))
= plus(2, plus(2, mult(1, 2)))
= plus(2, plus(2, plus(2, mult(0, 2))))
= plus(2, plus(2, plus(2, 0)))
= plus(2, plus(2, 2))
= plus(2, 4)
= 6
```

参考答案

```
def mult(m, n):
    if m == 0:
        return 0
    m1 = m - 1
    return plus(n, (mult(m1, n)))
```

错误示例

```
def mult(m, n):
    if m == 0:
        return n
    m1 = m - 1
    return plus(n, (mult(m1, n)))
```

mult是乘法，不是加法！

```
def mult(m, n):
    if m == 0:
        return 0
    m1 = m - 1
    return plus(n, (plus(m1, n)))
```

m*n=2n+m-1???

参考答案

```
def mult(m, n):
    if m == 0:
        return 0
    m1 = m - 1
    return plus(n, (mult(m1, n)))
```

错误示例

```
def mult(m, n):
    if m == 0:
        return 0
    m1 = m - 1
    return plus(n, (mult(m-1, n)))
```

参考答案

```
def mult(m, n):
    if m == 0:
        return 0
    m1 = m - 1
    return plus(n, (mult(m1, n)))
```

错误示例

```
def mult(m, n):
"""
mult(m, n) returns the value of m*n, where both m and n are positive integers.
Just put one name or constant in each of the blank space.
Do NOT write anything more.
"""
```

Problem 6

—— **Trees**（你们最爱的种树）

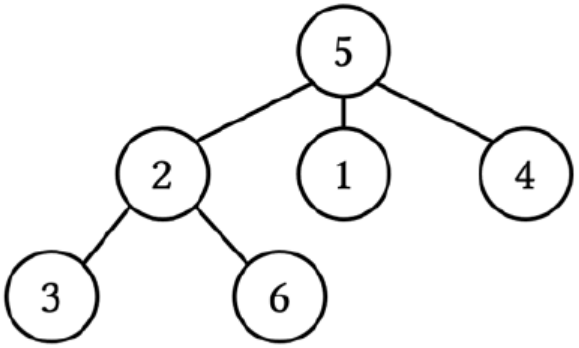
Same constructors and selectors as HW and LAB

```
def tree(label, branches=[]):
    return [label] + list(branches)

def label(t):
    return t[0]
```

```
def is_leaf(t);
    return not branches(t)

def branches(t):
    return t[1:]
```

build a tree

t1



tree(5, [tree(2. [tree(3), tree(6)]), tree(1), tree(4)])

tree(5, (tree(2. (tree(3), tree(6))), tree(1), tree(4)))

[5, [2, [3, 6], [1, 4]]]

tree(5, [tree(2, [3, 6]]), 1, 4])

```
def label_sum(t):
    if _________________ :
        return _________________
    return _______ + sum([_________________ for _______ in _____________________ ])
```

```
def label_sum(t):
    if is_leaf(t):
        return label(t)
    return label(t) + sum([label_sum(branch) for branch in branches(t)])
```

```
def reverse_tree(t):
    if _____ :
        return _____
    reversed_branches = _____
    return _____
```

```
def reverse_tree(t):
    if is_leaf(t):
        return t
    reversed_branches = [reverse_tree(branch) for branch in branches(t)][::-1]
    return tree(label(t), reversed_branches)
```

```
def reverse_tree(t):
    if is_leaf(t):
        return t
    reversed_branches = reversed(branches(t))
    return tree(label(t), [reverse_tree(branch) for branch in reversed_branches])
```

```
def reverse_tree(t):
    if is_leaf(t):
        return t
    reversed_branches = branches(t).reverse()
    return tree(label(t), [reverse_tree(branch) for branch in reversed_branches])
```

```
def reverse_tree(t):
    if is_leaf(t):
        return t
    reversed_branches = [reverse_tree(branches(t)[-i])
                         for i in range(len(branches(t)))]
    return tree(label(t), reversed_branches)
```

a = [1, 2, 3], a[-0] =?

```
def reverse_tree(t):
    if is_leaf(t):
        return t
    reversed_branches = [reverse_tree(branch)
                         for branch in branches(t)][:-1:-1]
    return tree(label(t), reversed_branches)
```

```
def reverse_tree(t):
    if is_leaf(t):
        return t
    reversed_branches = [reverse_tree(branches(t)[i])
                         for i in range(len(branches(t), 0, -1))]
    return tree(label(t), reversed_branches)
```

```
def fold_tree(t, base_func, merge_func):
    if is_leaf(t):
        return base_func(label(t))
    return merge_func(label(t),
                      [fold_tree(subtree, base_func, merge_func)
                       for subtree in branches(t)])
```

What are the input and output of base_func and merge_func

base_func : label(tree) -> result
merge_func : label(tree), list of result -> result

```
def label_sum(t):
    return fold_tree(t, lambda v: v, lambda v, vs: v + sum(vs))


def height(t):
    return fold_tree(t, lambda _: 1, lambda v, vs: 1 + max(vs))


def preorder(t):
    return fold_tree(t, lambda v: [v], lambda v, vs: reduce(add, [v], vs)
```

```
def label_sum(t):
    return fold_tree(t, lambda v: v, lambda v, vs: sum(vs))

def label_sum(t):
    return fold_tree(t, lambda v: label(v),
                     lambda v, vs: label(v) + sum(vs))

def label_sum(t):
    return fold_tree(t, lambda v: v, lambda v, vs: sum(v, vs))
```

```
>>> sum(1, [1,2,3])
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

```
def height(t):
    return fold_tree(t, lambda _: 1, lambda v, vs: max(vs))
```

```
def height(t):
    return fold_tree(t, lambda _: 1, lambda v, vs: max(1, vs))
```

```
>>> max(1, [1,2,3])
[1, 2, 3]
```

```
def preorder(t):
    return fold_tree(t, lambda v: print(v),
                     lambda v, vs: …)
```

```
def preorder(t):
    return fold_tree(t, lambda v: [v],
                     lambda v, vs: reduce(lambda x, y: [x] + y, v, vs))
```

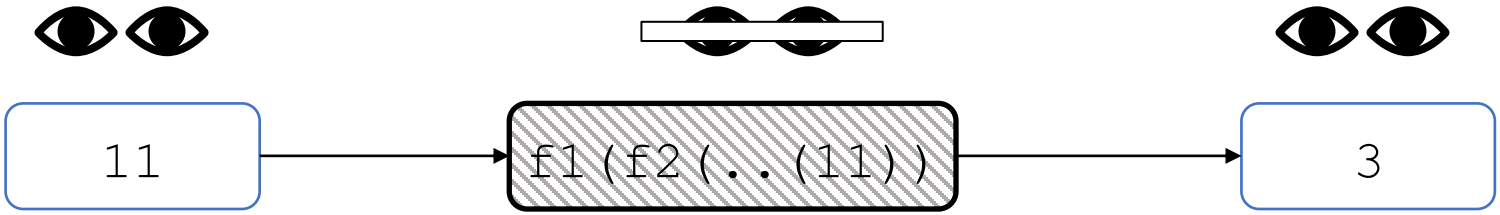
```
def preorder(t):
    return fold_tree(t, lambda v: [v], sum)
```

```
def preorder(t):
    return fold_tree(t, lambda v: [v], lambda v, vs: sum([v], vs))
```

Problem 7

—— Automatic Function Composition

问题理解



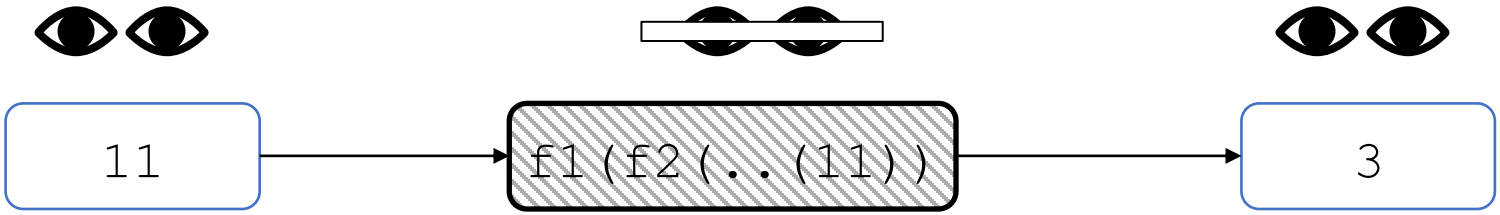
输入

选取一些函数，先复合，
然后喂入输入11，看看
算出来是不是3

输出

函数从哪儿来？
哪些函数可以选呢？

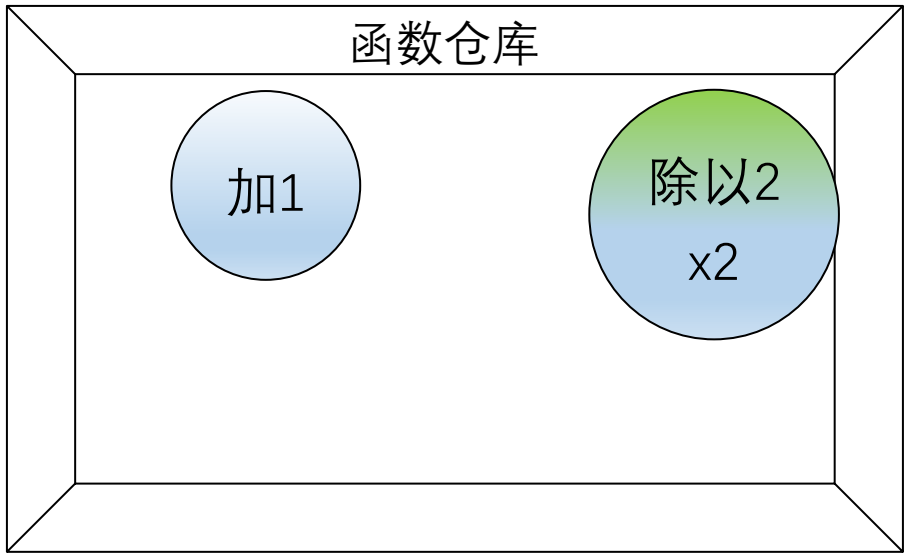
问题理解



输入

选取**仓库**里的一些函数，把它们复合，然后喂入输入11，看看算出来是不是3

输出



有1个inc（加1）

有2个div2（除以2）

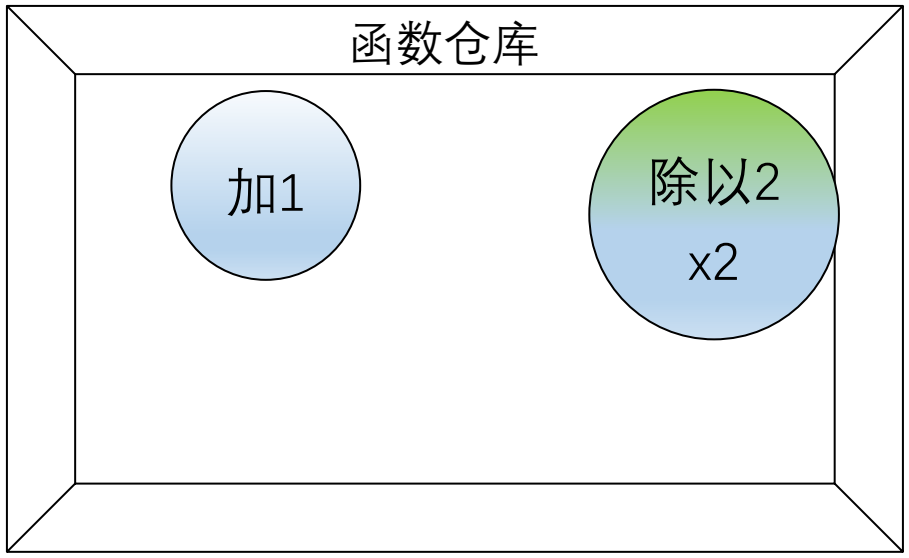
怎么做?

11

输入

```
inc(11)                 div2(inc(11))
inc(div2(11))           div2(inc(div2(11)))
inc(div2(div2(11)       div2(div2(11))
div2(11)                div2(div2(inc(11)))
```

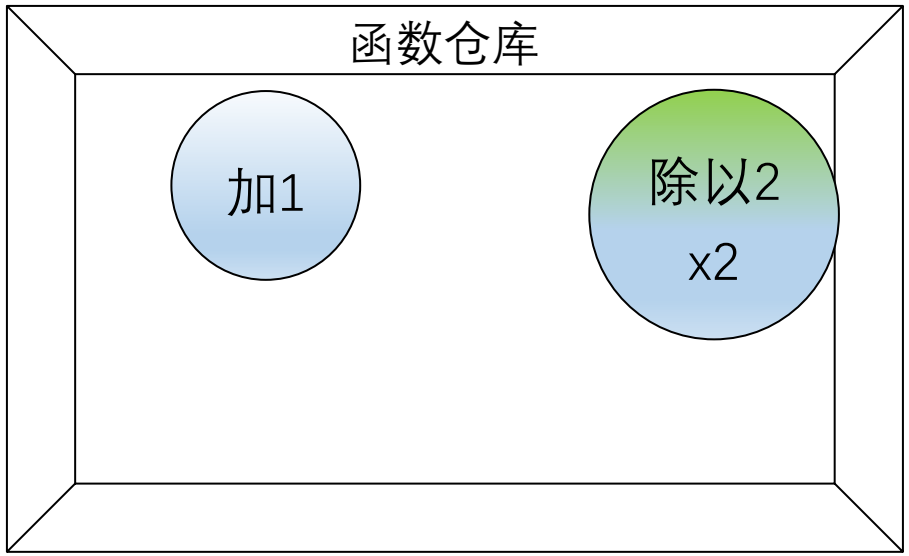
管他三七二十一，列举仓库里面所有的函数组合，然后把输入11作为实参喂进去试试呗!



有1个inc（加1）

有2个div2（除以2）

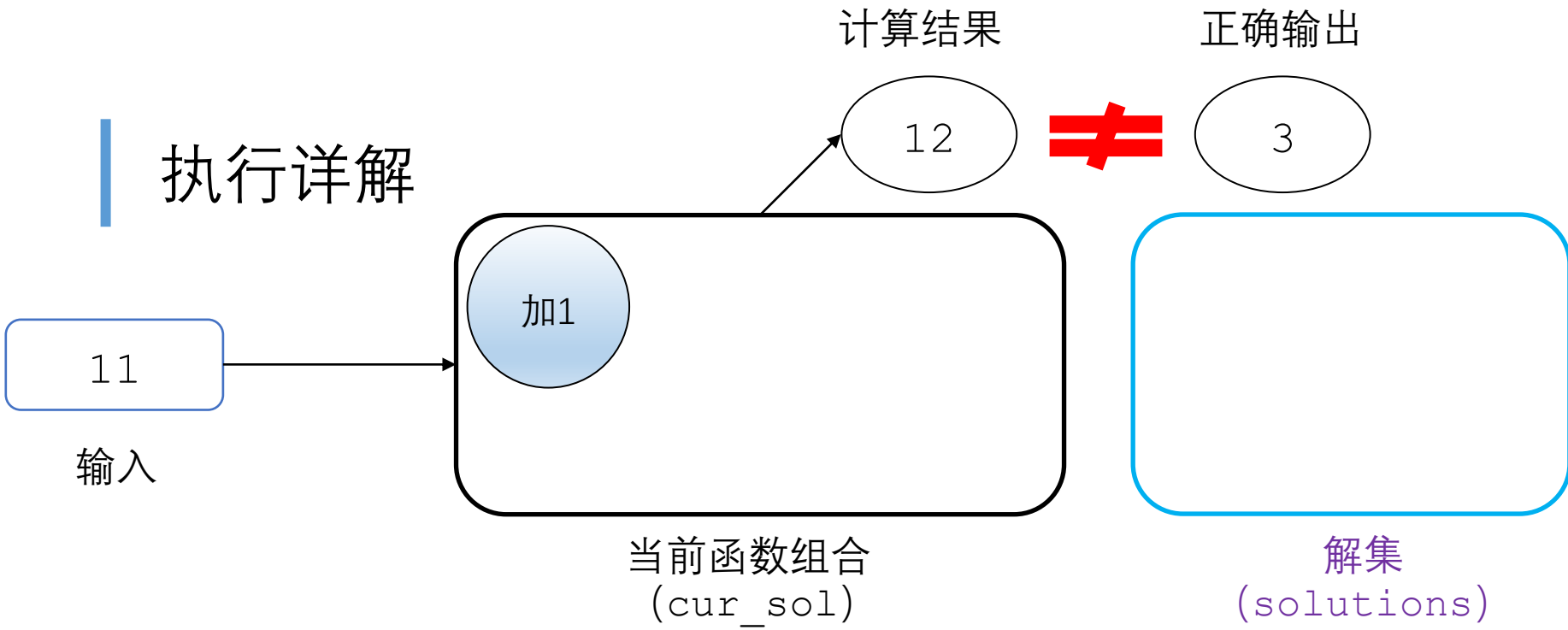
执行详解

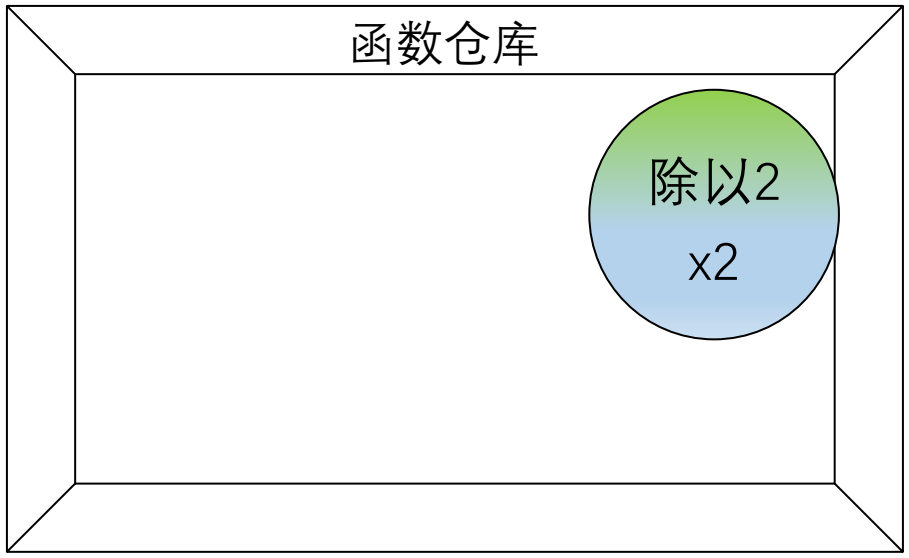


还剩1个inc（加1）

还剩2个div2（除以2）

执行详解

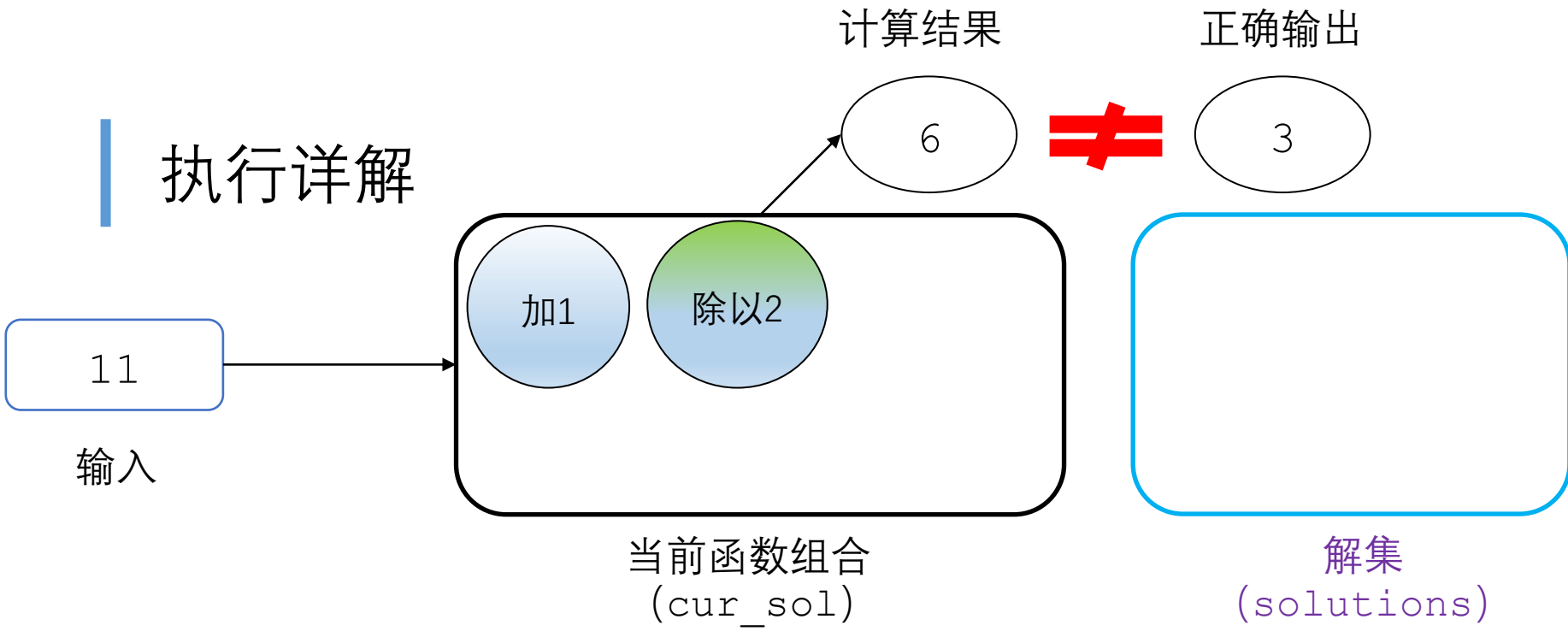


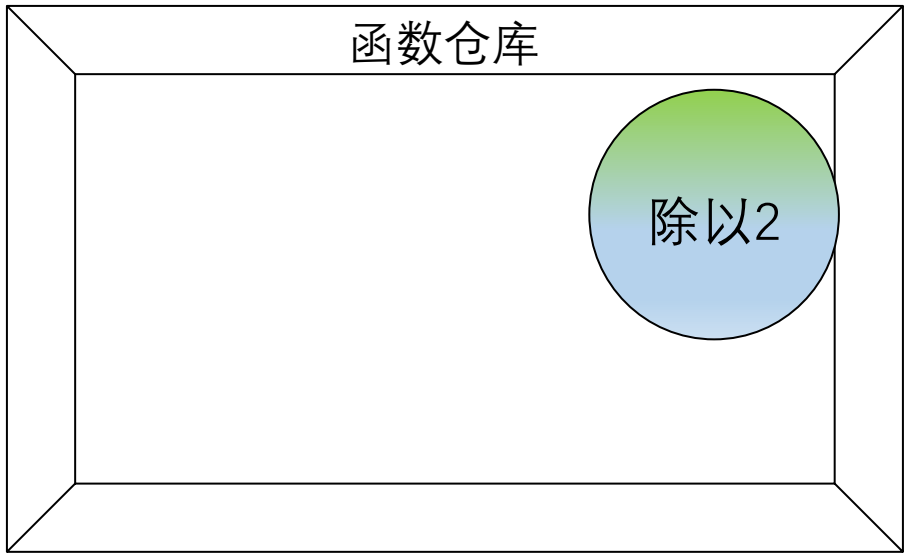


还剩0个inc（加1）

还剩2个div2（除以2）

执行详解

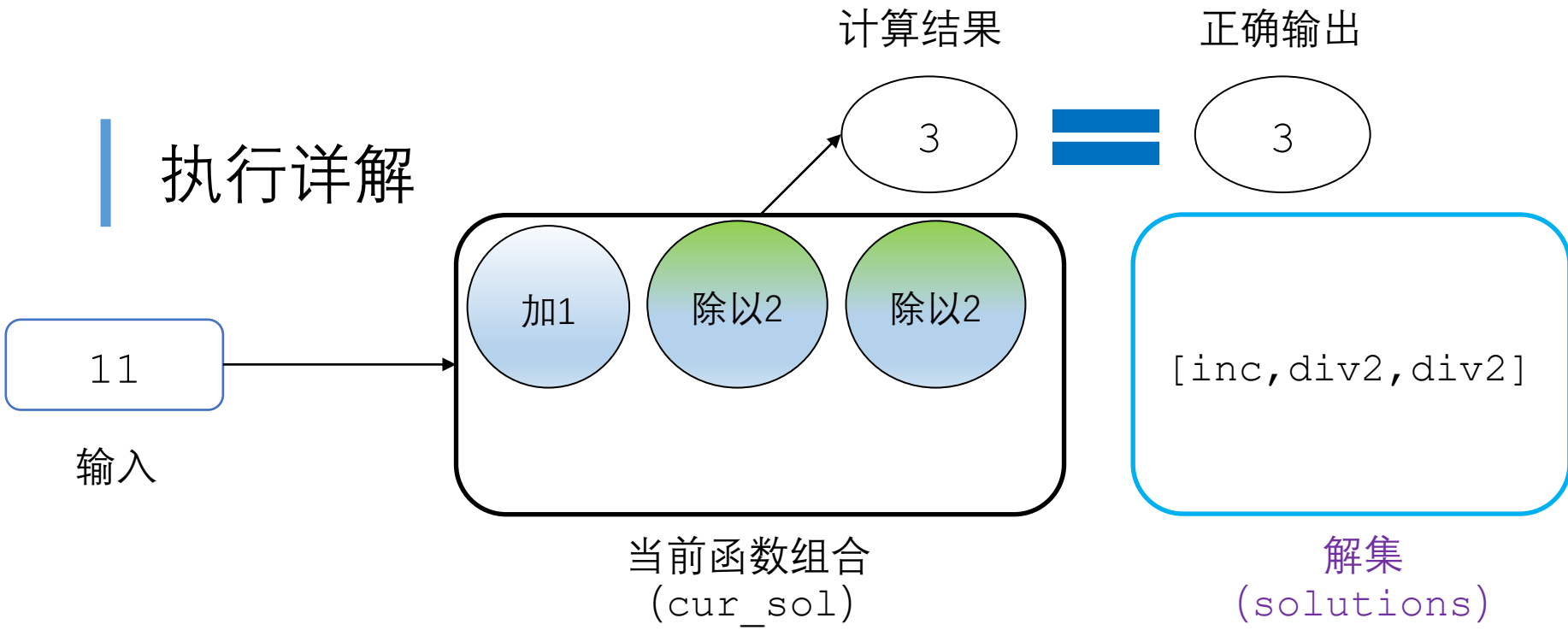


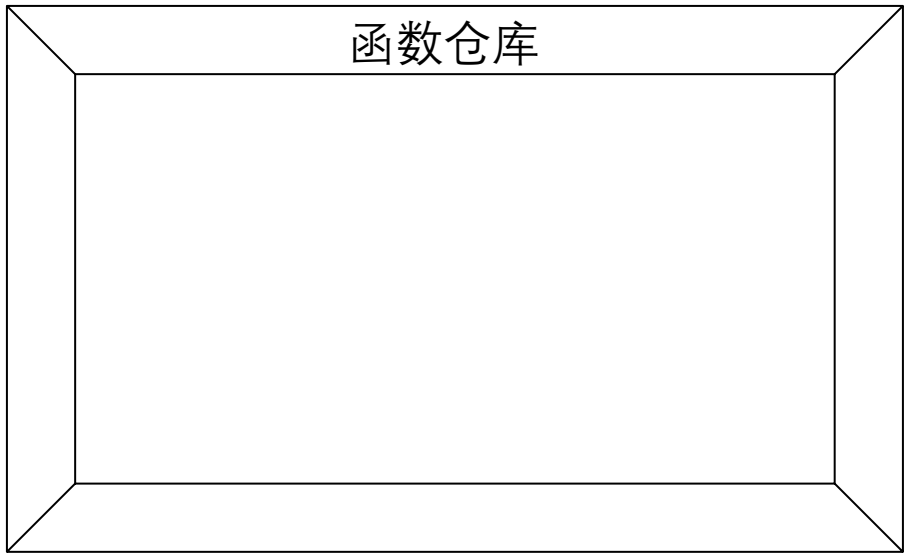


还剩0个inc（加1）

还剩1个div2（除以2）

执行详解



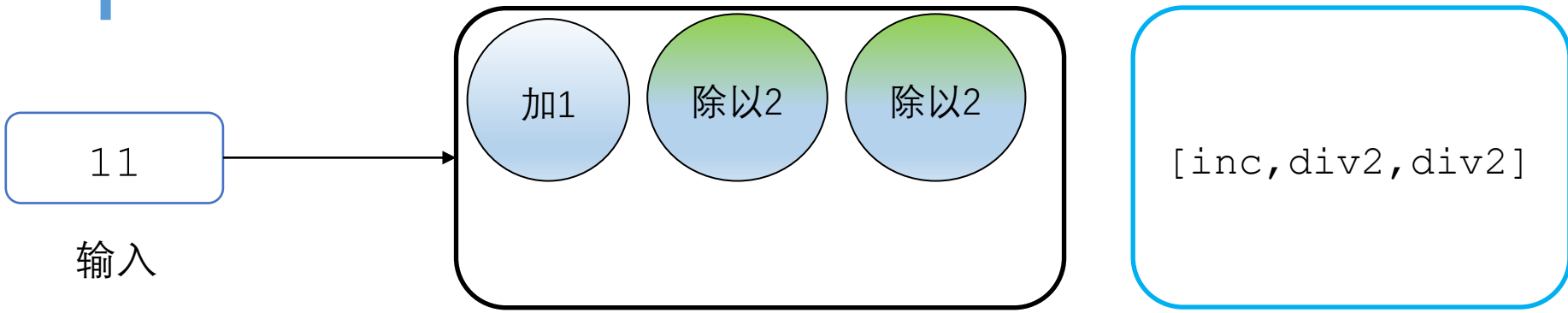


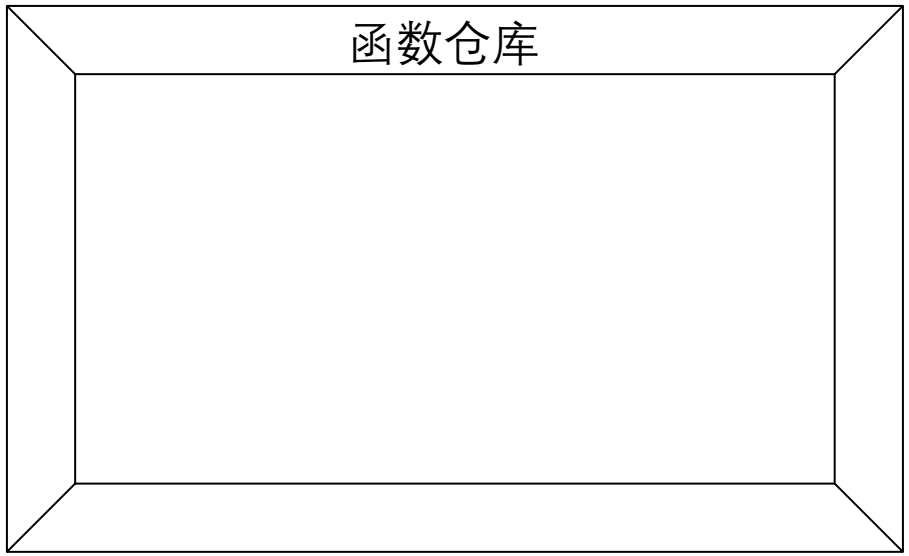
还剩0个inc（加1）

还剩0个div2（除以2）

执行详解

仓库里没有能用的函数了，把函数放回去，找找新的组合!



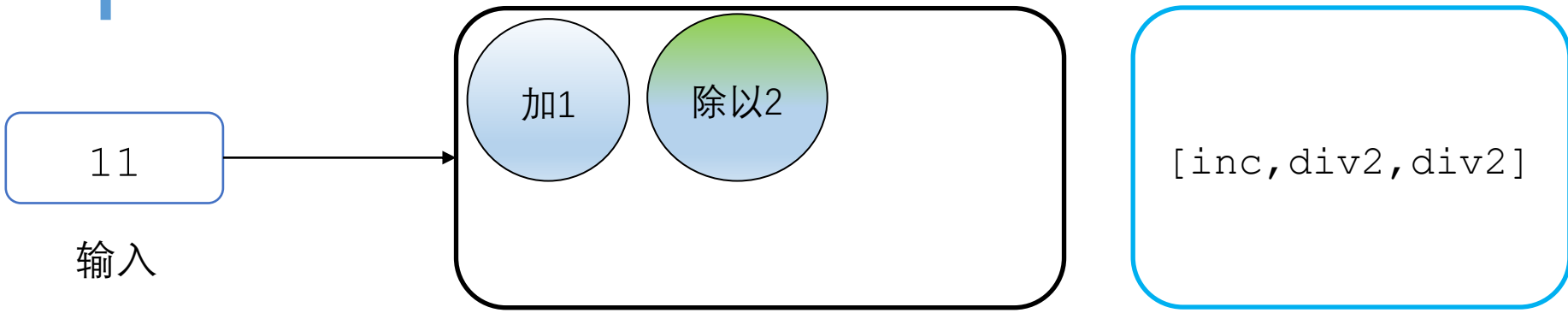


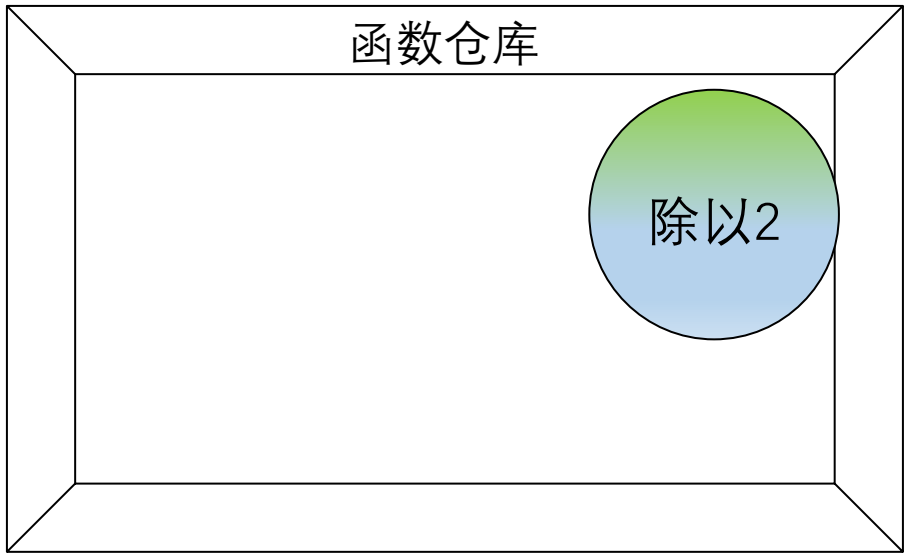
还剩0个inc（加1）

还剩0个div2（除以2）

执行详解

这时只能使用仓库里的除以2，然而刚刚已经试过，那么继续回溯



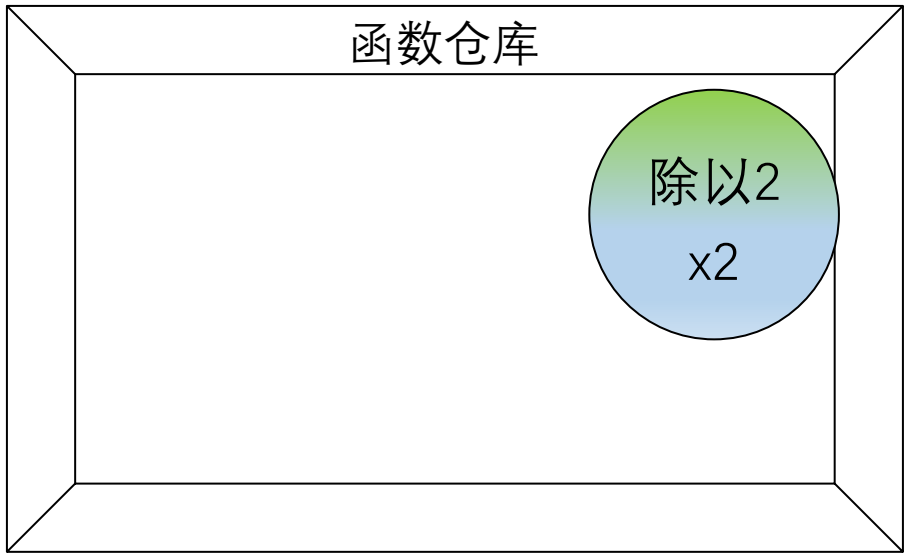


还剩0个inc（加1）

还剩1个div2（除以2）

执行详解

这时只能使用仓库里的除以2，然而刚刚已经试过，那么继续回溯

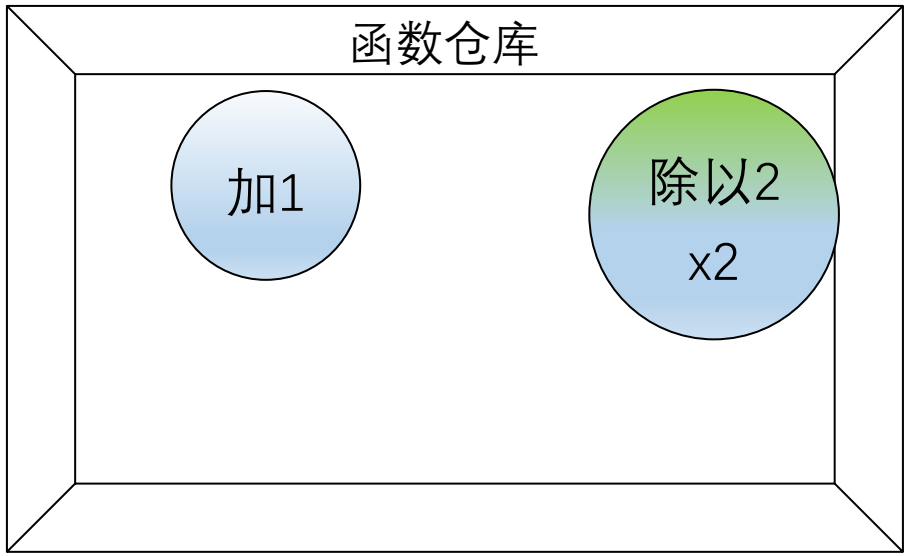


还剩0个inc（加1）

还剩2个div2（除以2）

执行详解

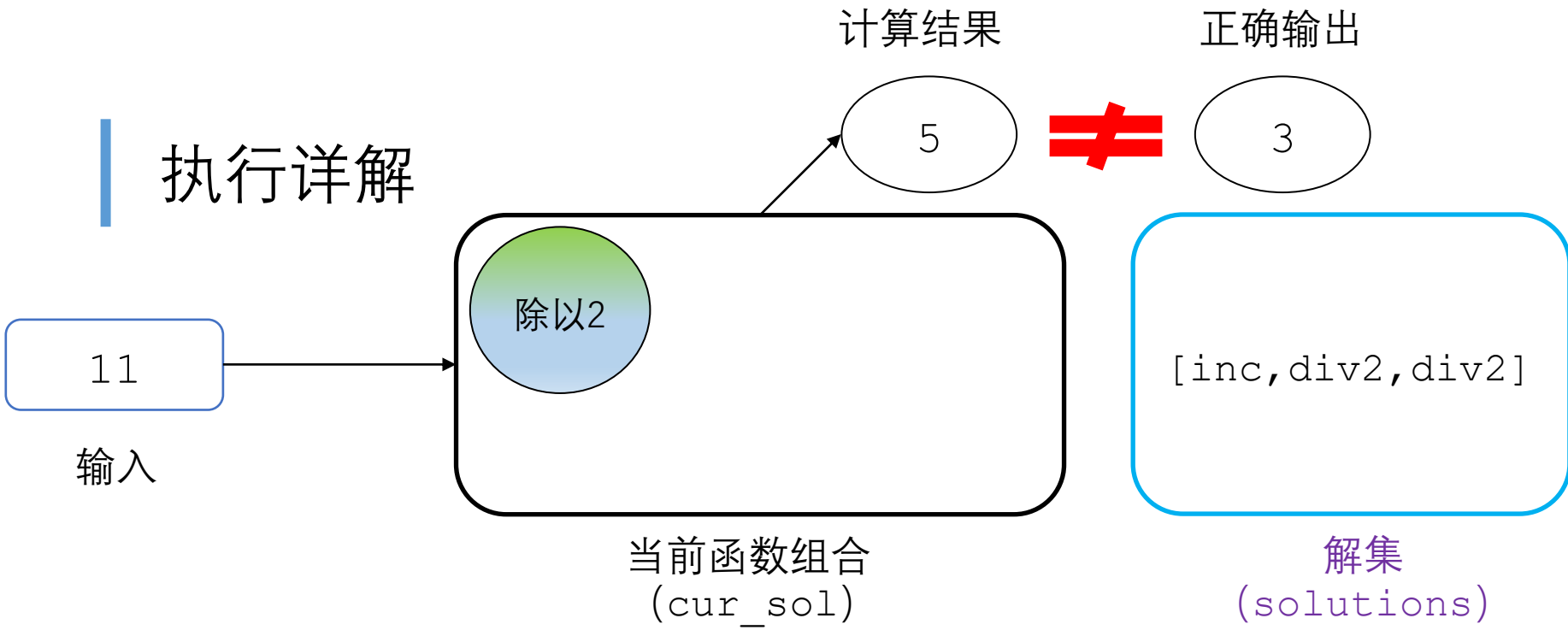
刚刚试过了先放加1，现在先把除以2放进去试试看!

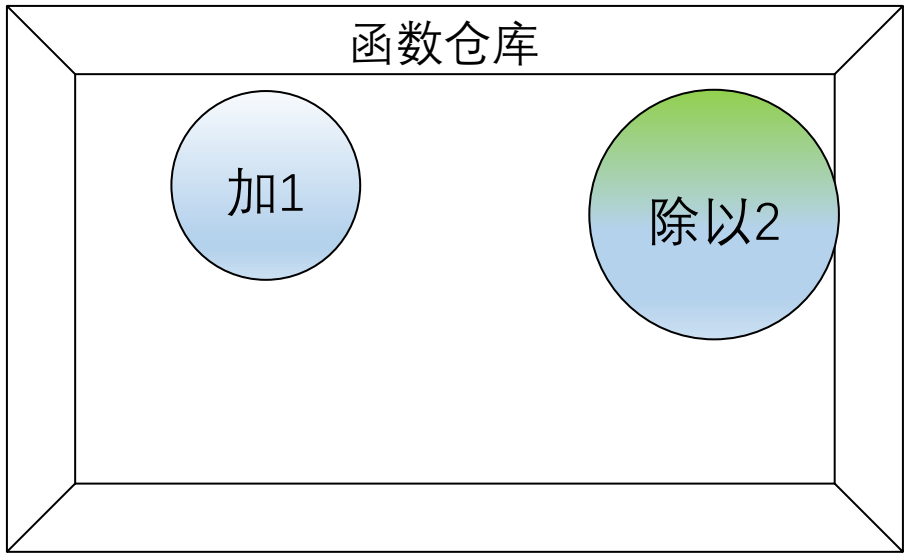


还剩1个inc（加1）

还剩2个div2（除以2）

执行详解

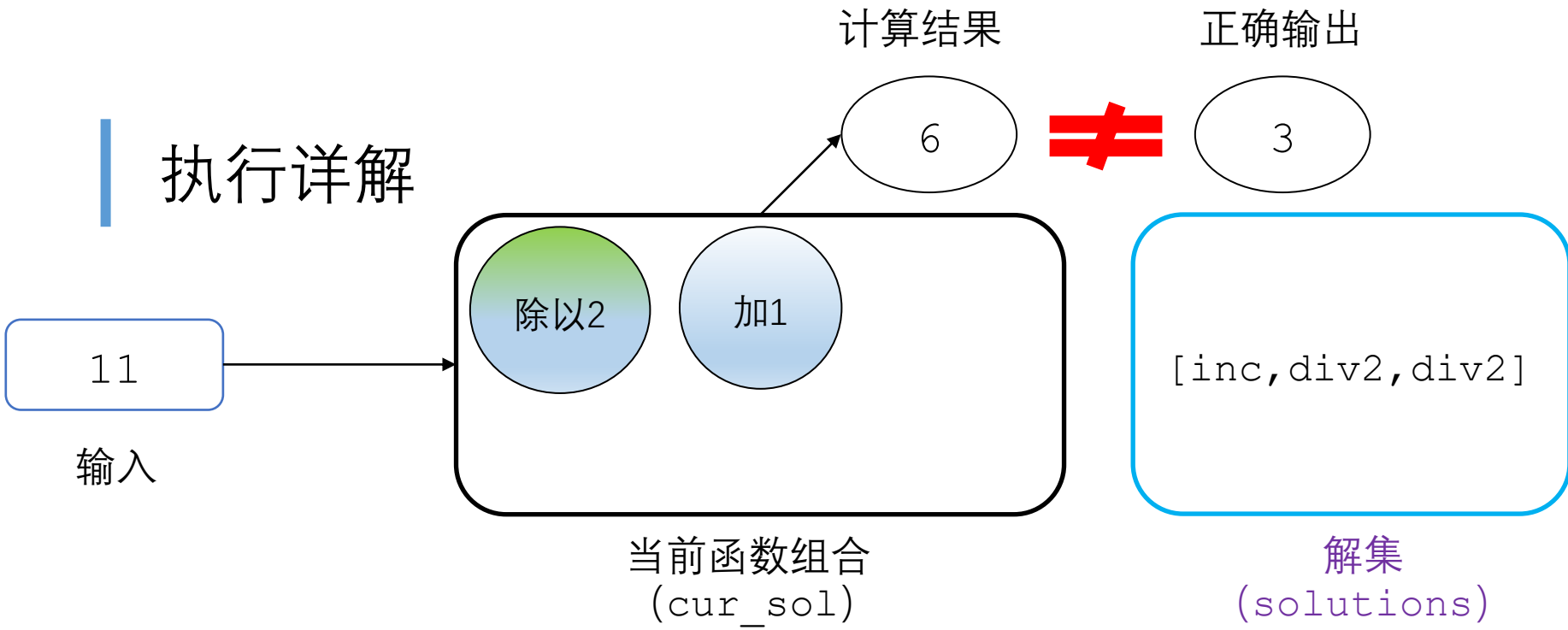


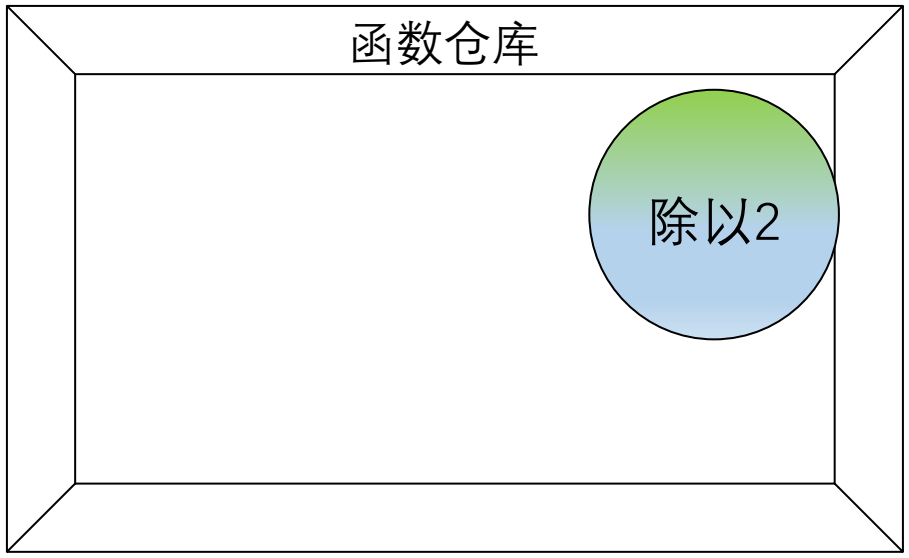


还剩1个inc（加1）

还剩1个div2（除以2）

执行详解

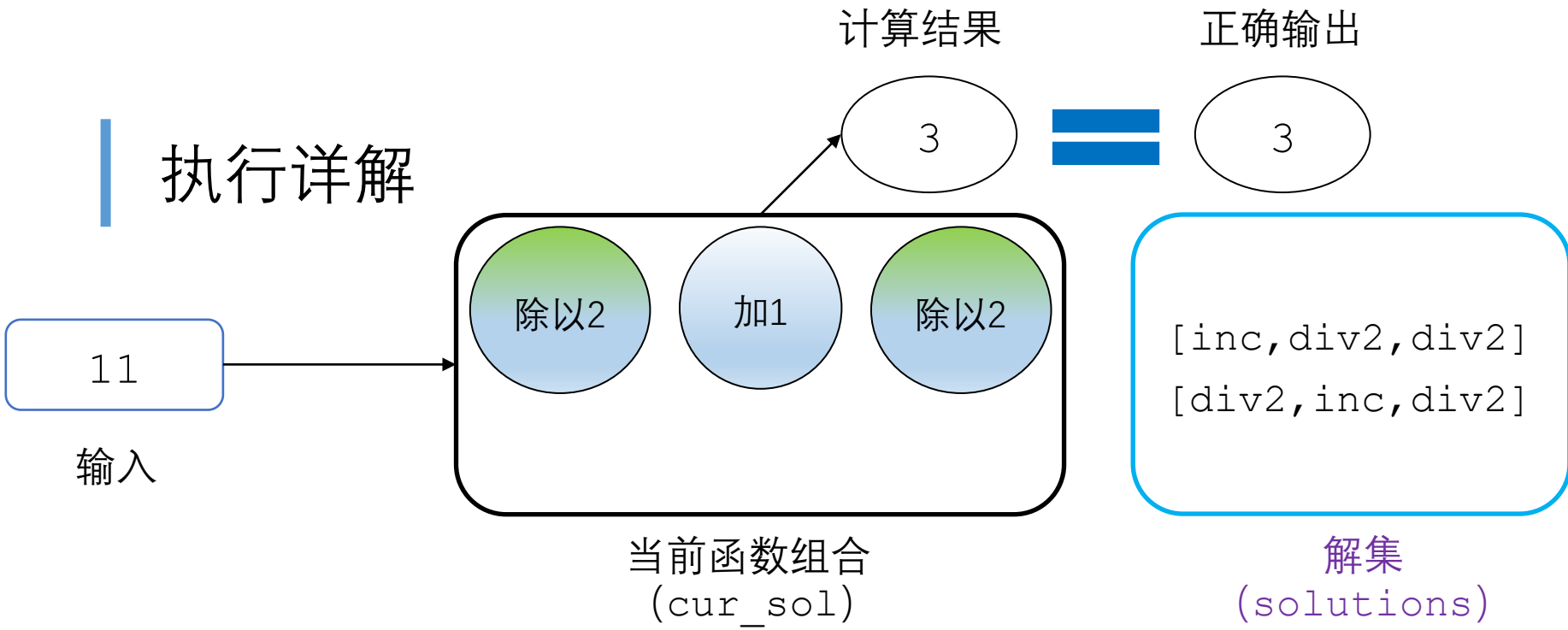


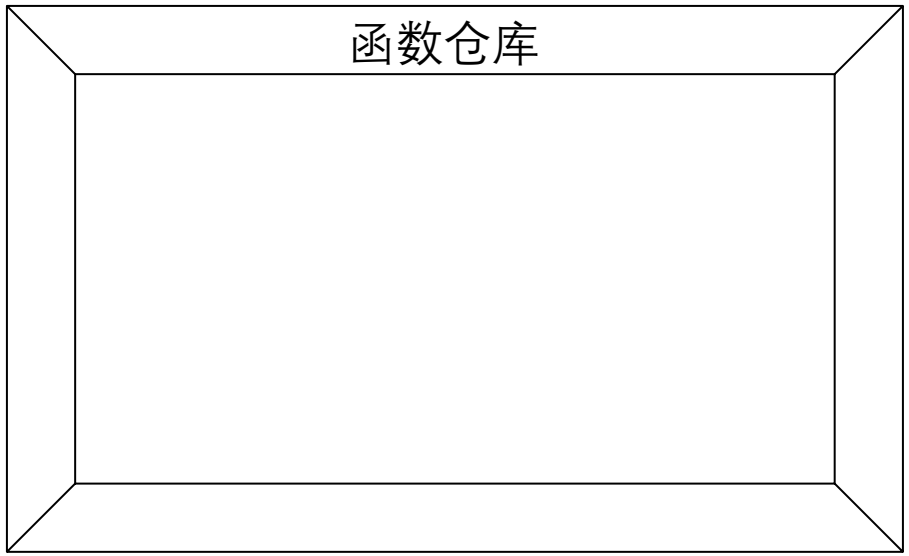


还剩0个inc（加1）

还剩1个div2（除以2）

执行详解



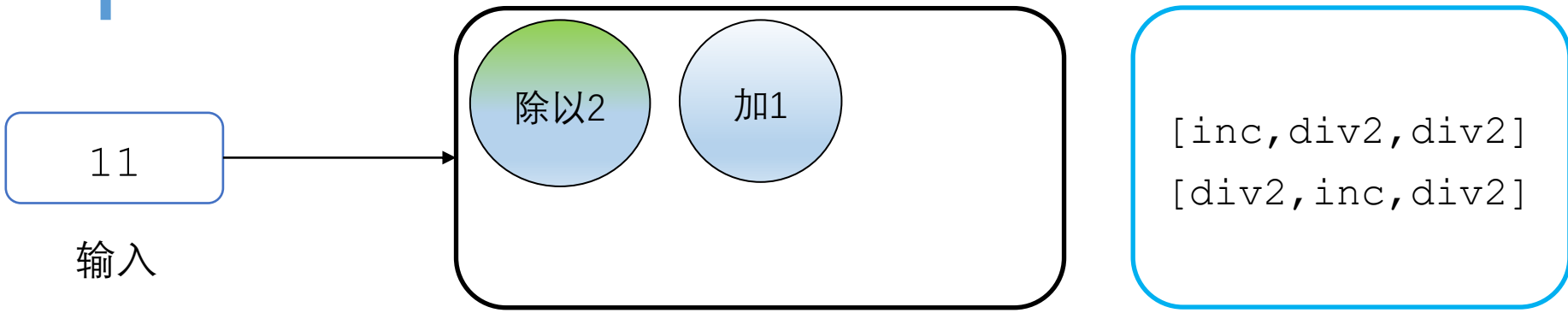


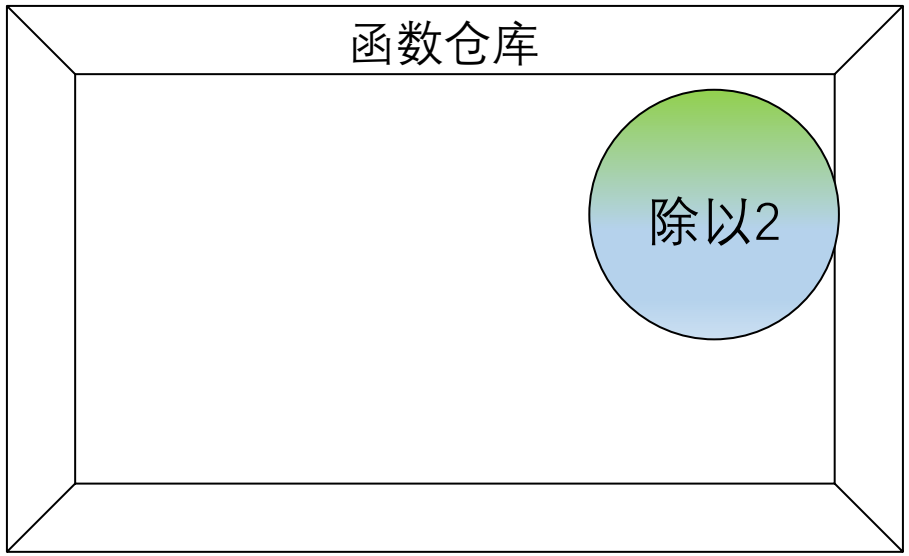
还剩0个inc（加1）

还剩0个div2（除以2）

执行详解

这时只能使用仓库里的除以2，然而刚刚已经试过，那么继续回溯



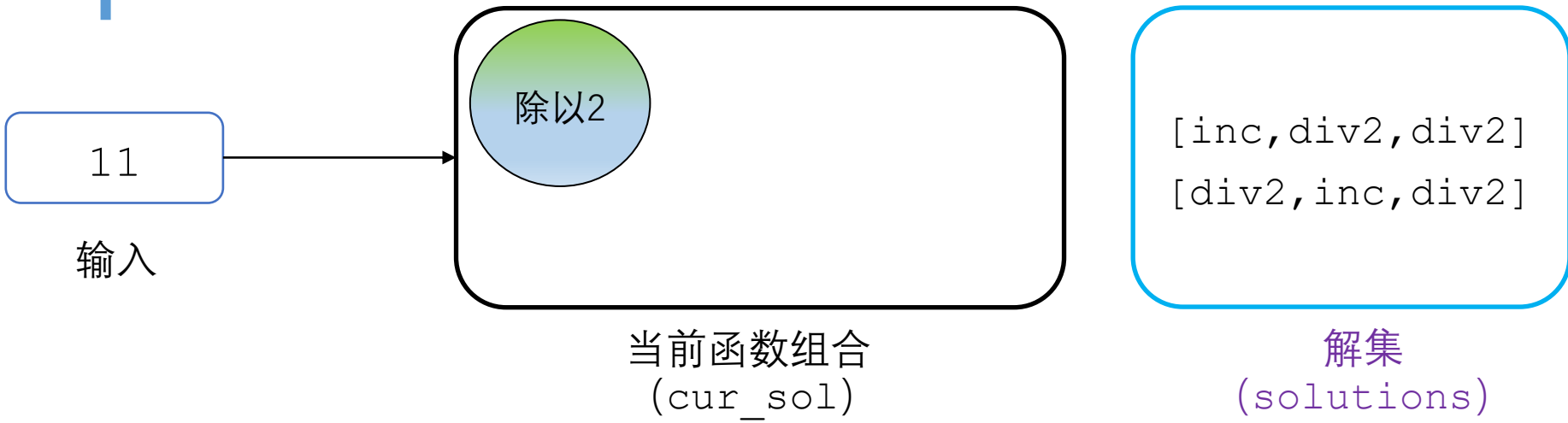


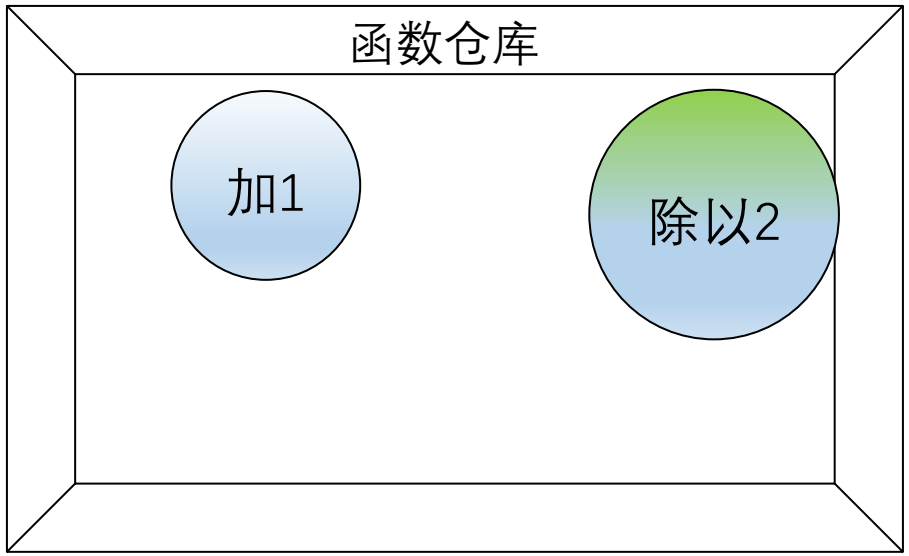
还剩0个inc（加1）

还剩1个div2（除以2）

执行详解

刚刚先放了加1，现在试着先放除以2

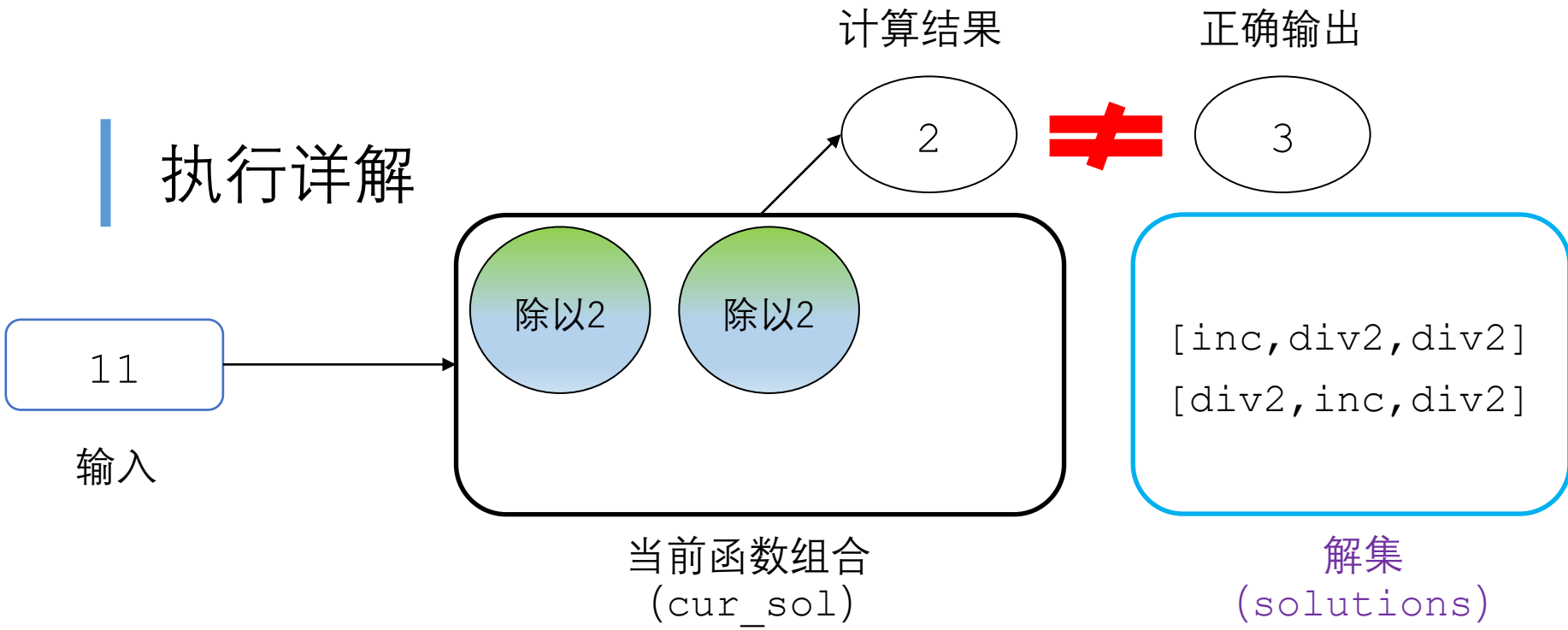


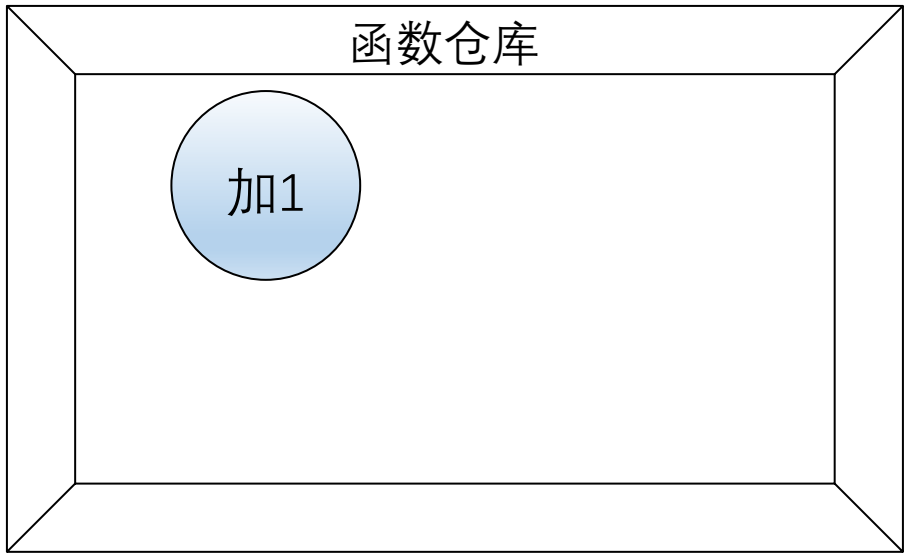


还剩1个inc（加1）

还剩1个div2（除以2）

执行详解

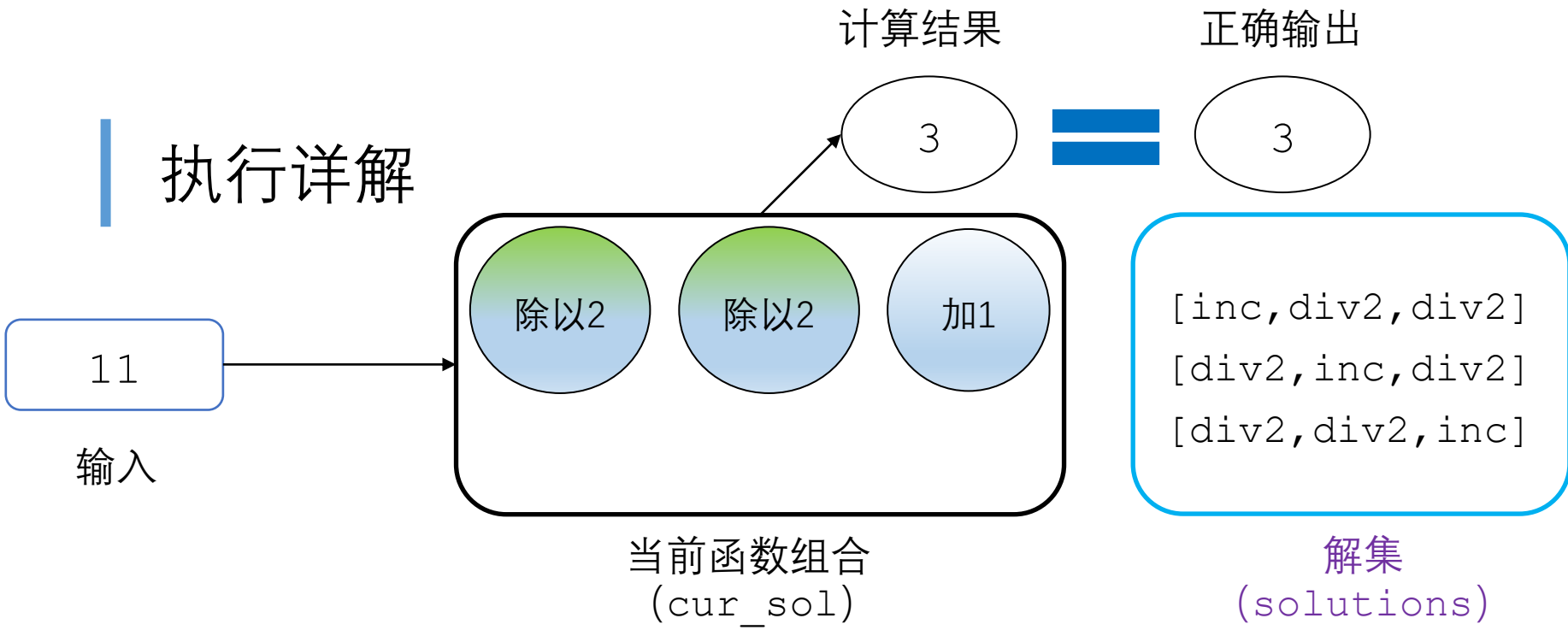


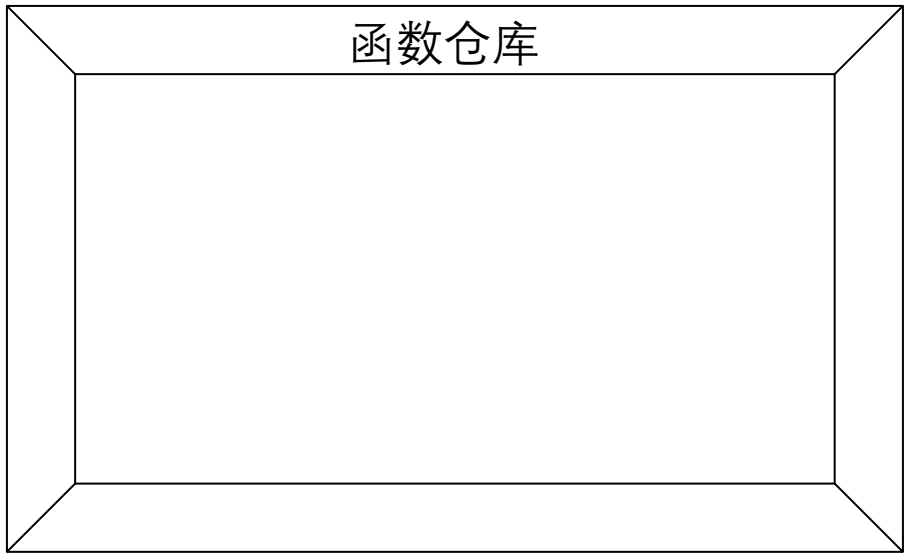


还剩1个inc（加1）

还剩0个div2（除以2）

执行详解



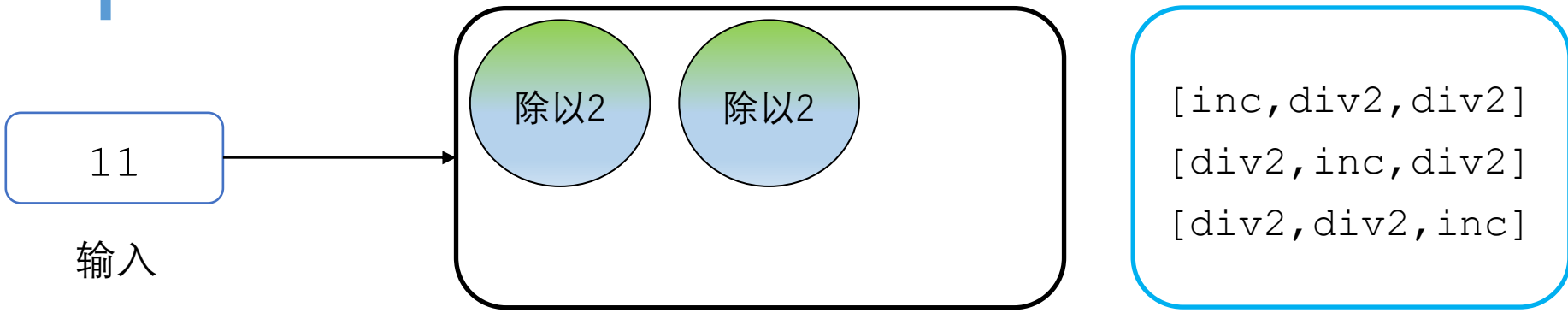


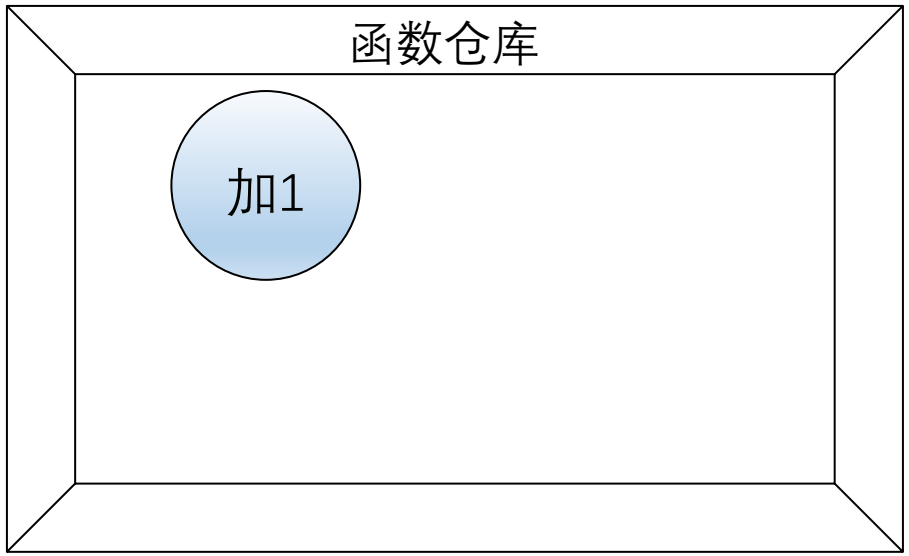
还剩0个inc（加1）

还剩0个div2（除以2）

执行详解

这时只能使用仓库里的加1，然而
刚刚已经试过，那么继续回溯



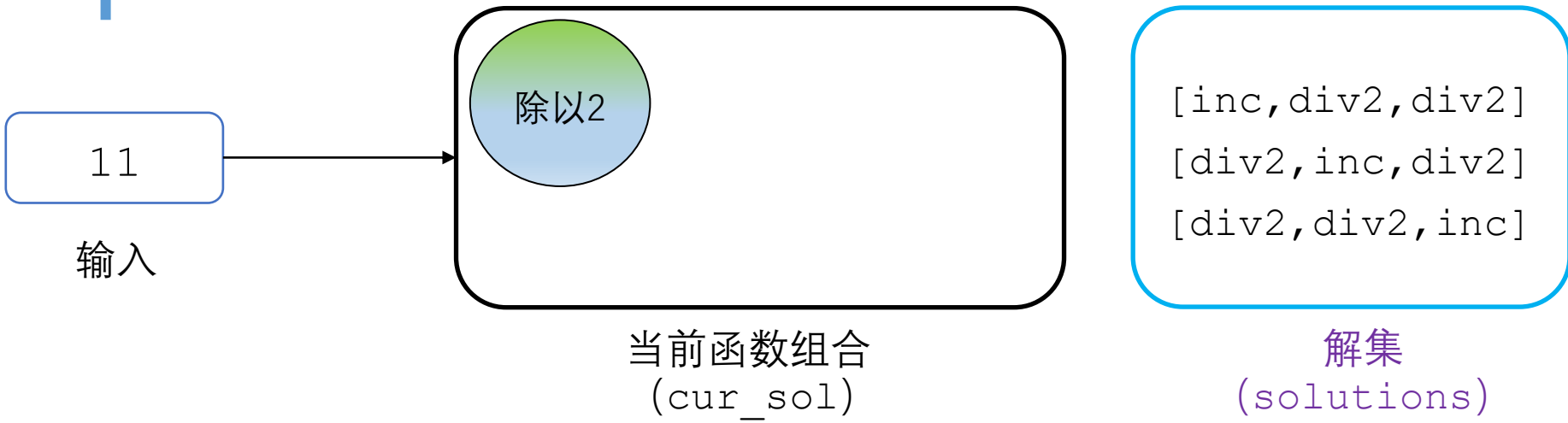


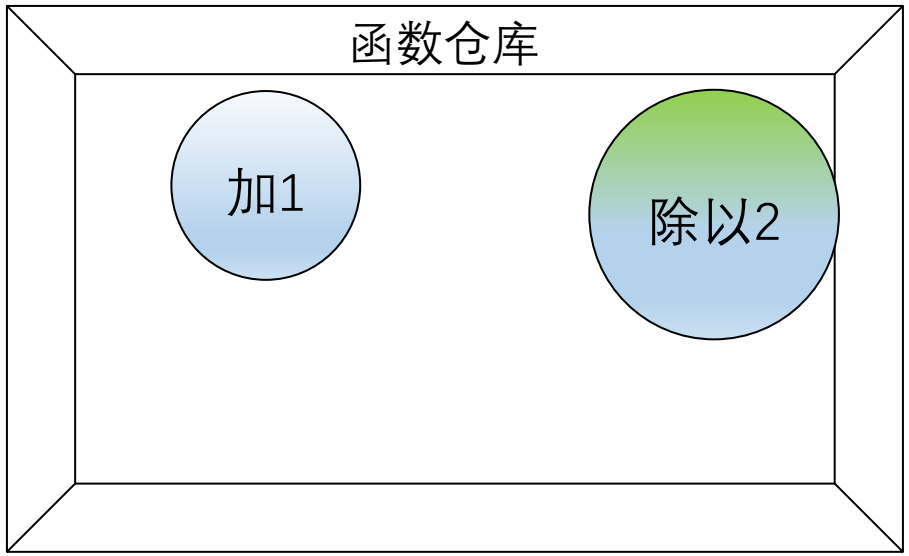
还剩1个inc（加1）

还剩0个div2（除以2）

执行详解

先放加1和先放除以2都已经试过了，继续回溯



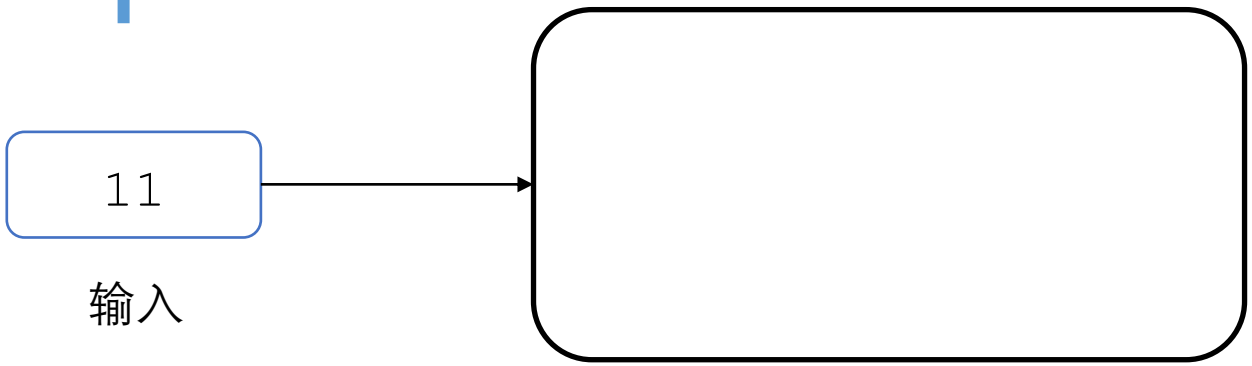


还剩1个inc（加1）

还剩1个div2（除以2）

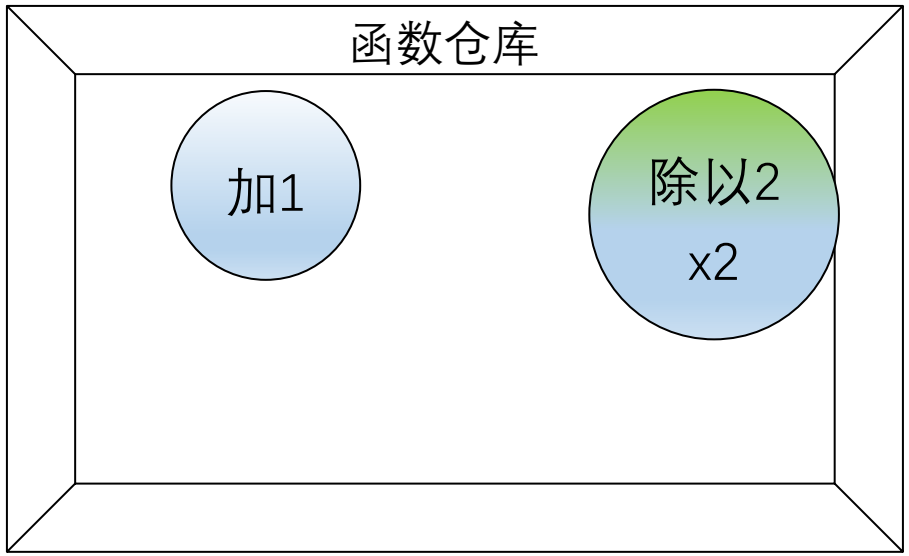
执行详解

先放加1和先放除以2都已经试过了，也回溯不了了，搜索结束!



[inc,div2,div2]
[div2,inc,div2]
[div2,div2,inc]

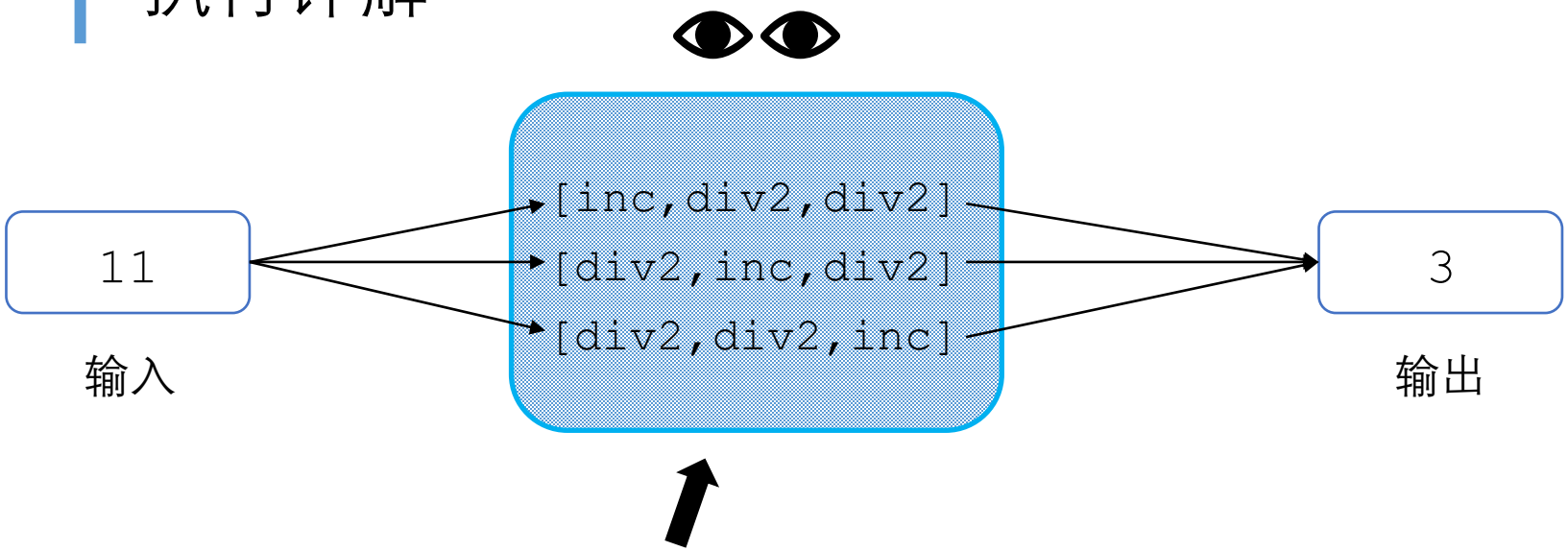
解集
(solutions)

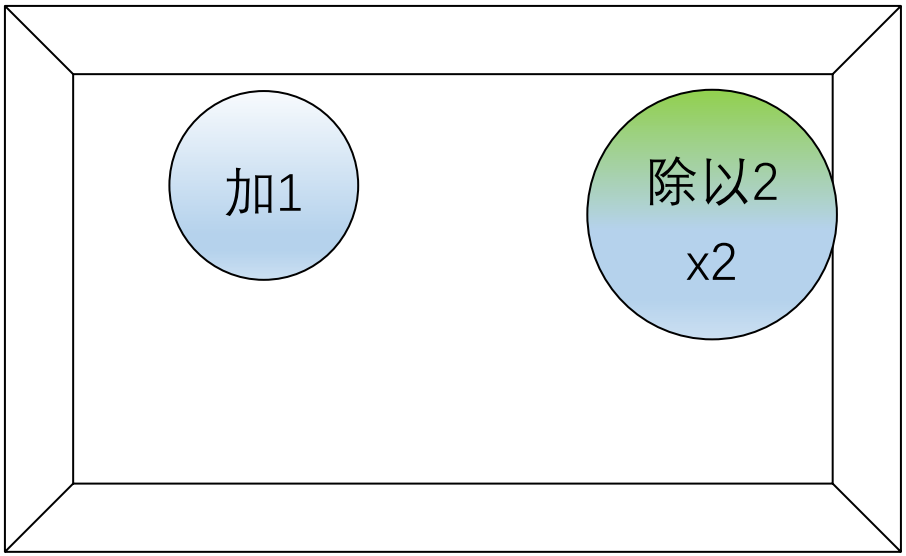


还剩1个inc（加1）

还剩2个div2（除以2）

执行详解

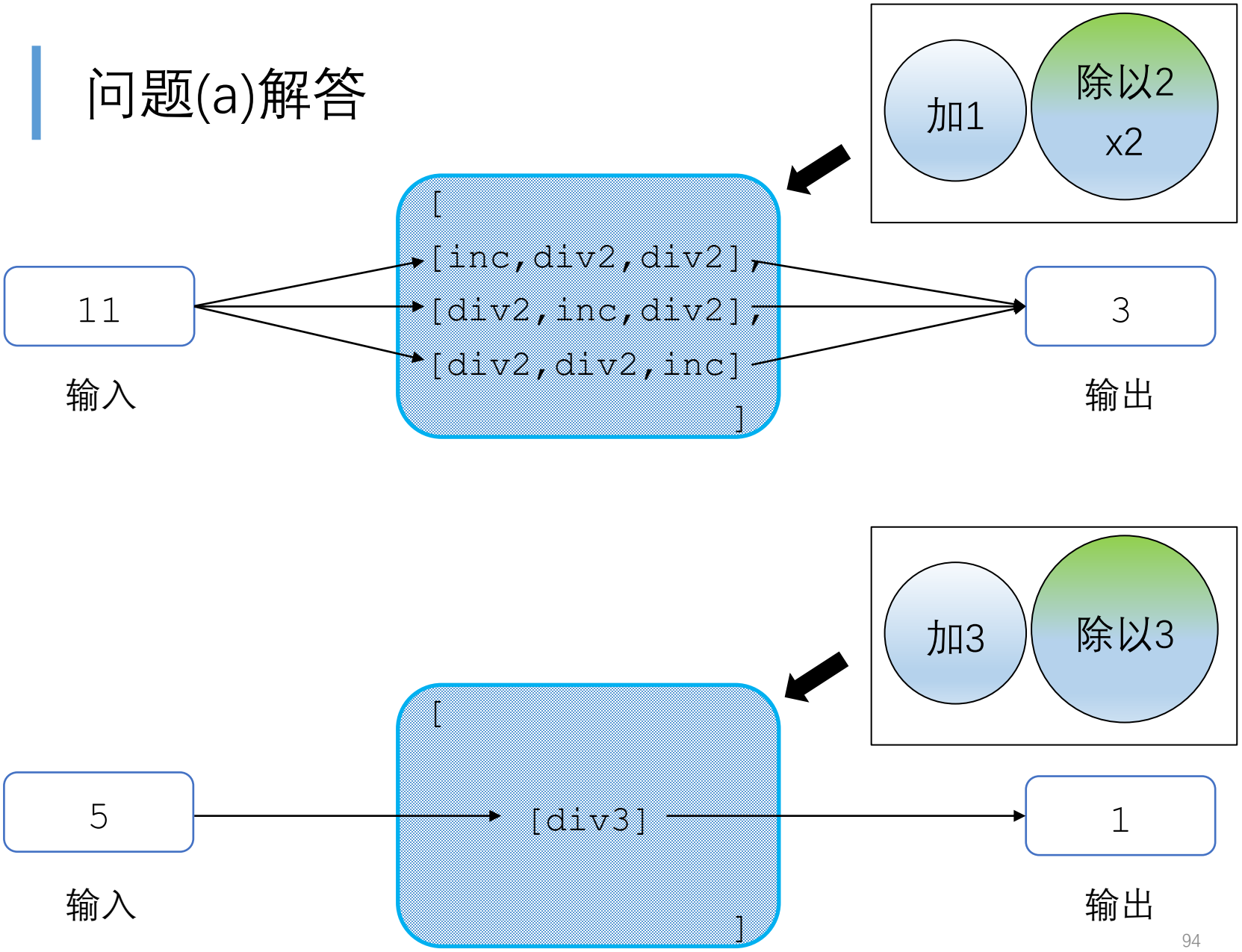




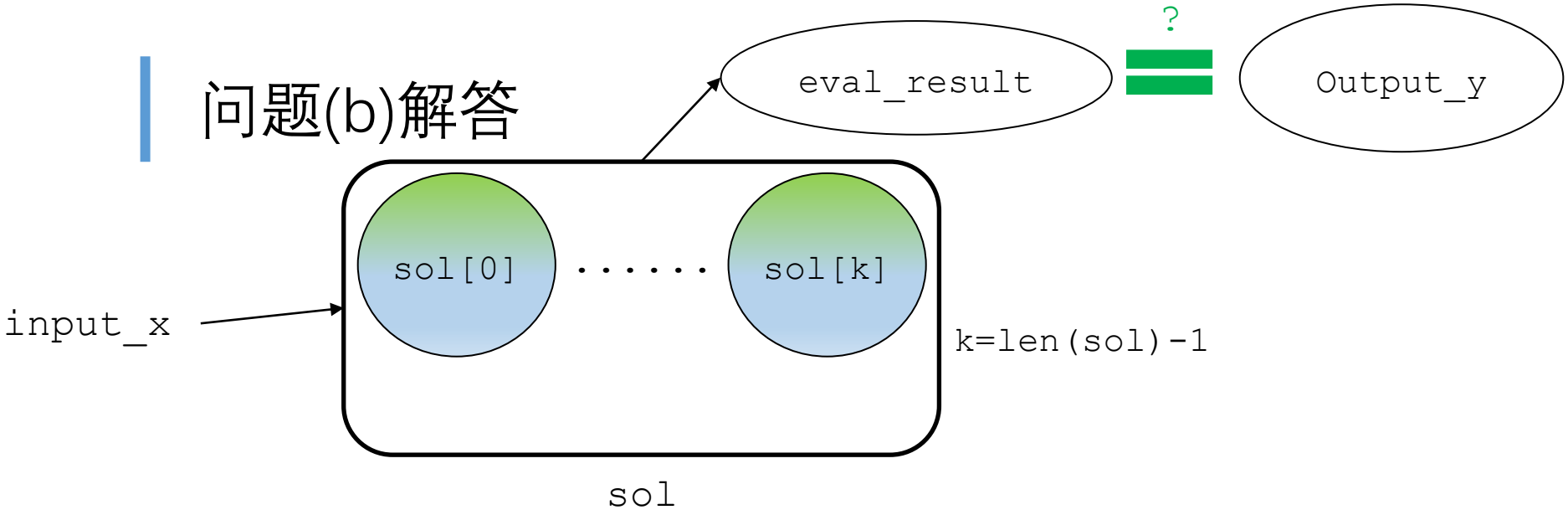
有1个inc（加1）

有2个div2（除以2）

问题(a)解答


加1
除以2
x2
[
[inc,div2,div2],
[div2,inc,div2],
[div2,div2,inc]
]
11
输入
3
输出
加3
除以3
[
[div3]
]
5
输入
1
输出

问题(b)解答



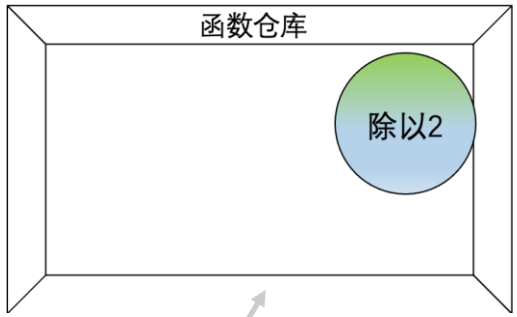

```
def satisfy(io_pair, sol):
    input_x = io_pair[0]
    output_y = io_pair[1]
    eval_result = input_x
    for func in sol:
        eval_result = func(eval_result)
    return eval_result == output_y
```

每空1分

问题(b)解答

```
def afc(io_pair, func_list, times_list):
    assert len(func_list) == len(times_list)
    solutions = []
    def sol_search(remain_times_list, cur_sol):
        if cur_sol and satisfy(io_pair, cur_sol):
            solutions.append(cur_sol)
        for i in range(len(func_list)):
            if remain_times_list[i] > 0:
                new_times_list = [remain_times_list[j] \
                  if j != i else remain_times_list[j] - 1
                  for j in range(len(func_list))]
                sol_search(new_times_list, cur_sol \
                  + [func_list[i]])
    sol_search(times_list, [])
    return solutions
```

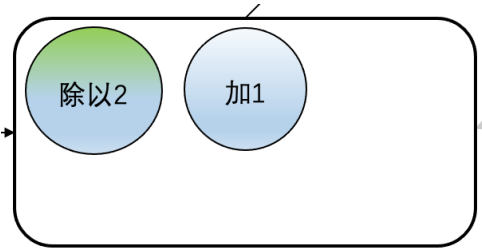
问题(b)解答



还剩0个inc（加1）

还剩1个div2（除以2）

```
def afc(io_pair, func_list, times_list):
    assert len(func_list) == len(times_list)
    solutions = []
    def sol_search(remain_times_list, cur_sol):
        if cur_sol and satisfy(io_pair, cur_sol):
            solutions.append(cur_sol)
        for i in range(len(func_list)):
            if remain_times_list[i] > 0:
                new_times_list = [remain_times_list[j] \
                  if j != i else remain_times_list[j] - 1
                  for j in range(len(func_list))]
                sol_search(new_times_list, cur_sol \
                  + [func_list[i]])
    sol_search(times_list, [])
    return solutions
```



当前函数组合
(cur_sol)

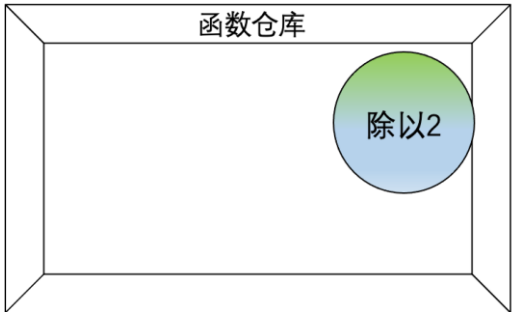
问题(b)解答

Goal: We want to automatically find a non-empty list of m functions $[f_{i_1}, f_{i_2}, \ldots, f_{i_m}]$

```
# Every solution should NOT be empty
solutions = []
def sol_search(remain_times_list, cur_sol):
    if cur_sol and satisfy(io_pair, cur_sol):
        solutions.append(cur_sol)
    for i in range(len(func_list)):
        if remain_times_list[i] > 0:
            new_times_list = [remain_times_list[j] \
              if j != i else remain_times_list[j] - 1
              for j in range(len(func_list))]
            sol_search(new_times_list, cur_sol \
              + [func_list[i]])
sol_search(times_list, [])
return solutions
```

两个条件各1分

问题(b)解答




```
def afc(io_pair, func_list, times_list):
    assert len(func_list) == len(times_list)
    solutions = []
    def sol_search(remain_times_list, cur_sol):
        if cur_sol and satisfy(io_pair, cur_sol):
            solutions.append(cur_sol)
        for i in range(len(func_list)):
            if remain_times_list[i] > 0:
                new_times_list = [remain_times_list[j] \
                  if j != i else remain_times_list[j] - 1
                  for j in range(len(func_list))]
                sol_search(new_times_list, cur_sol \
                  + [func_list[i]])
    sol_search(times_list, [])
    return solutions
```

1分，f in func_list亦可

1分

times_list ✗

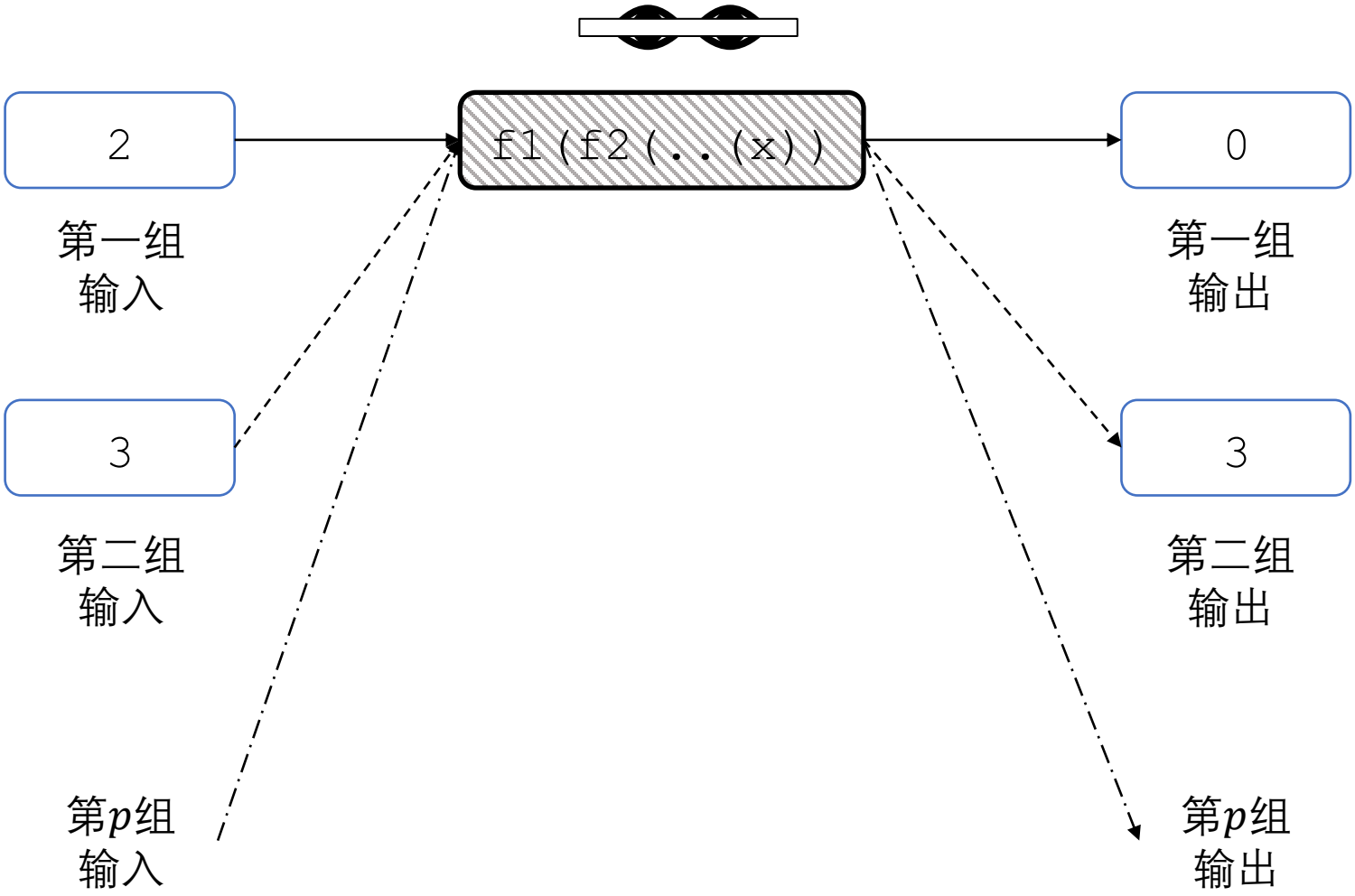
2分

2分

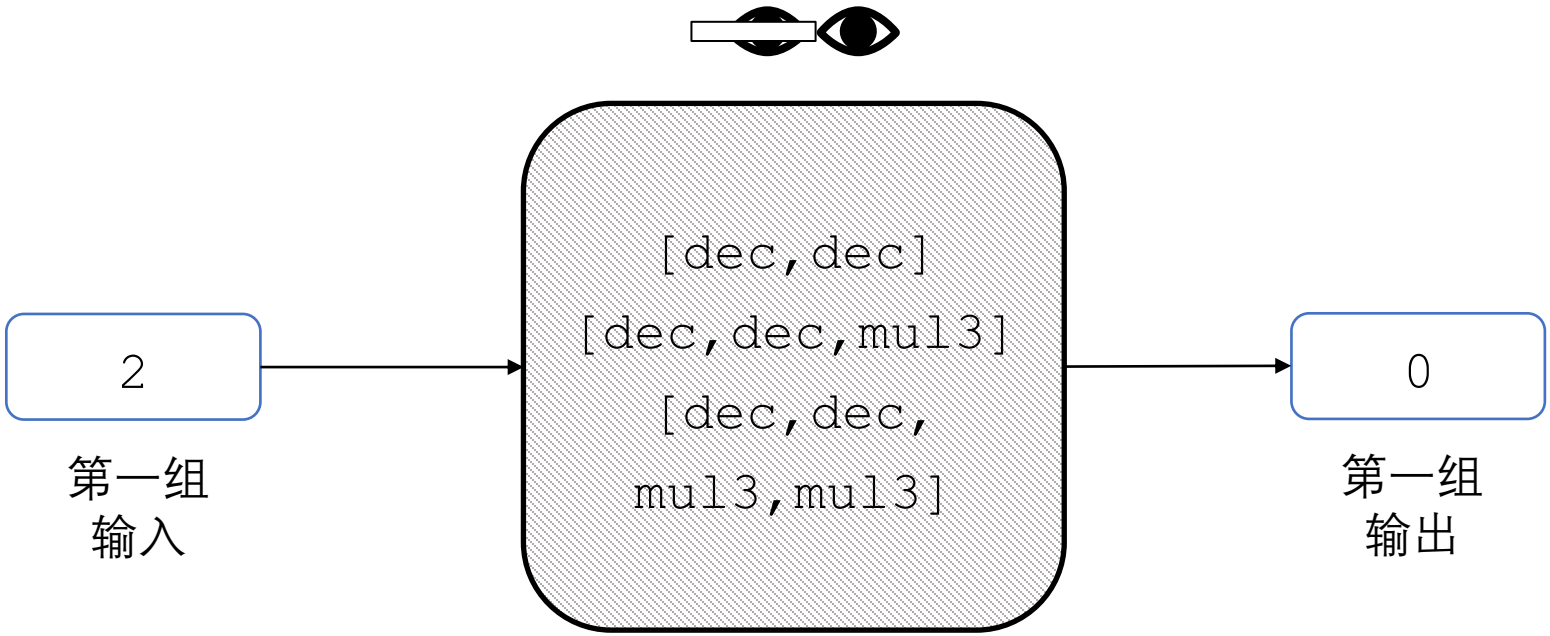
✗ [func_list[i]] + cur_sol

✗ cur_sol.append(func_list[i])

问题(c)



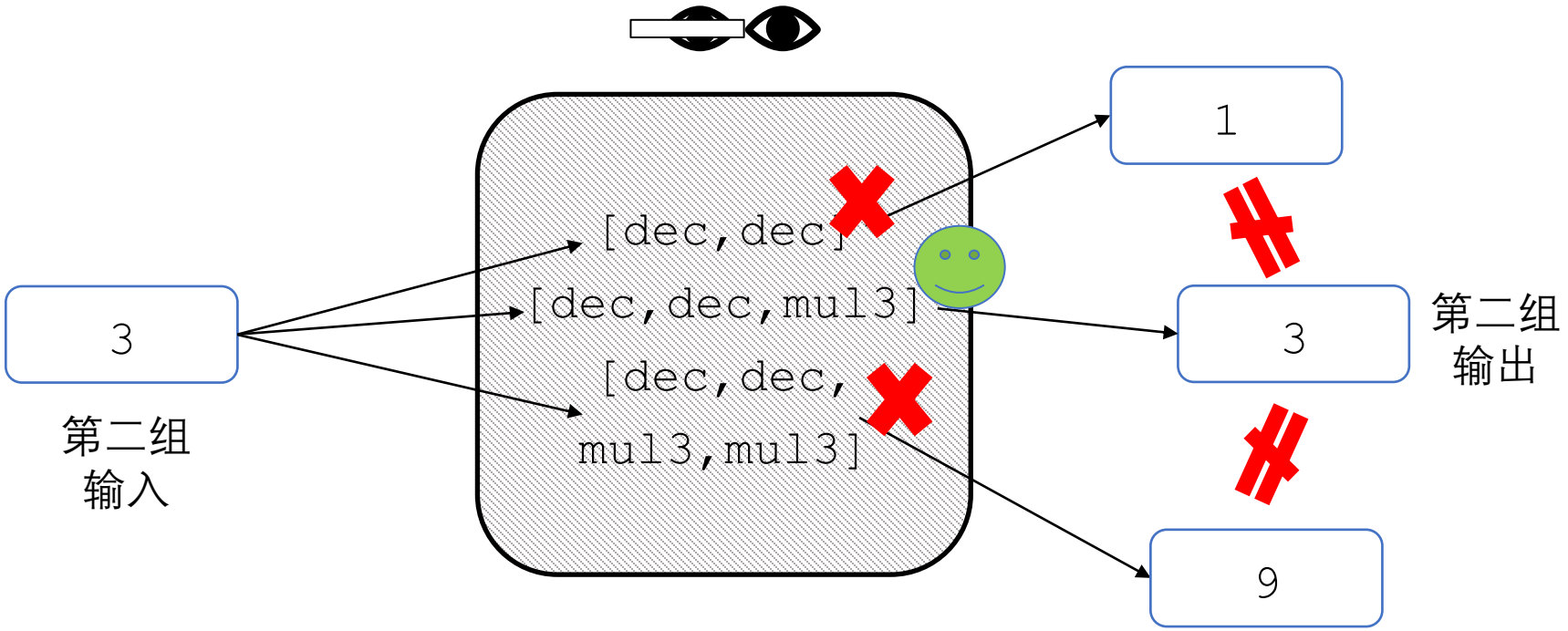
问题(c)理解



第一步

先将一组输入输出喂到前面的afc问题里面，算出解集。

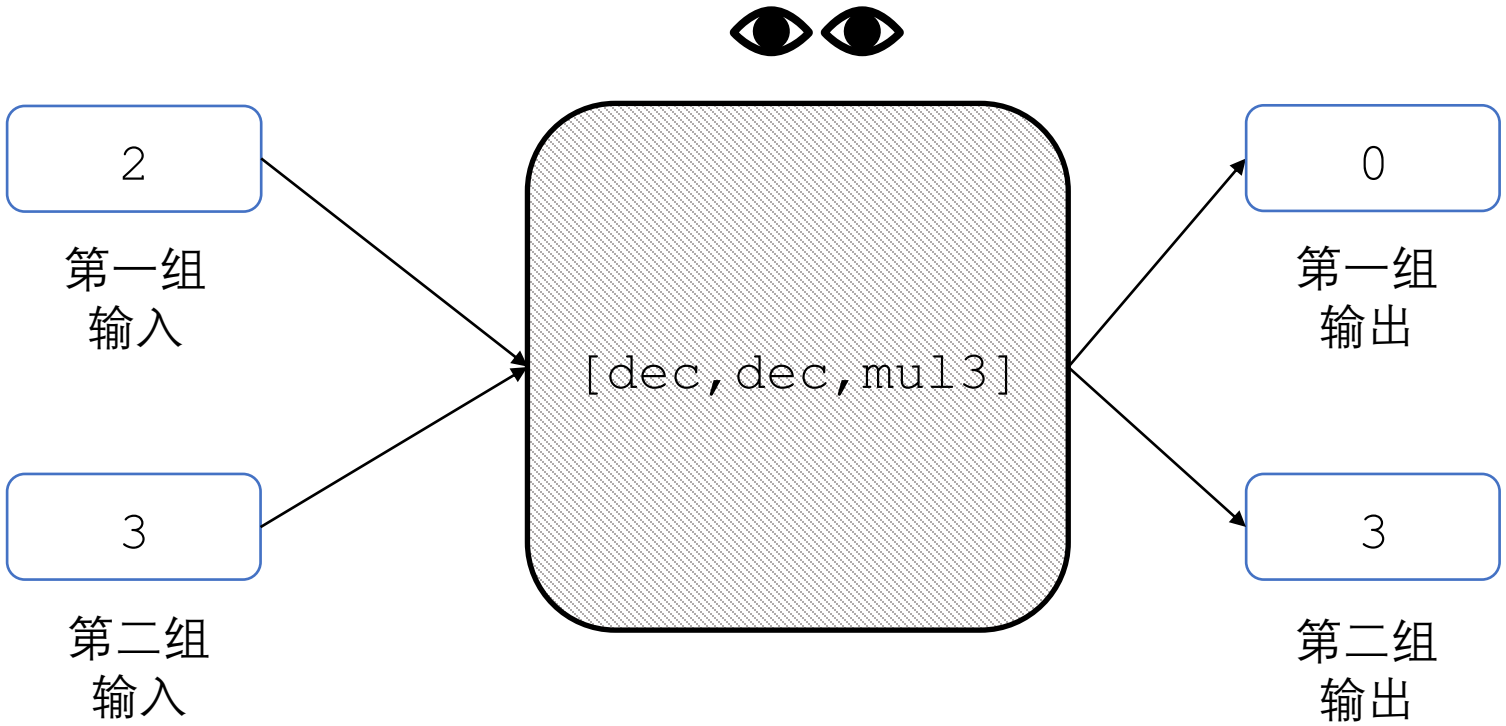
问题(c)理解



第二步

验证第一步算出来的每一个解是否满足所有输入输出?

问题(c)理解



问题(c)解答

```
def extended_afc(io_pair_list, func_list, times_list):
    assert len(io_pair_list) > 0
    solutions = afc(io_pair_list[0], func_list, times_list)
    return my_filter(lambda sol: my_all([satisfy(io_pair, sol) \
      for io_pair in io_pair_list], solutions)
```

问题(c)解答

```
def extended_afc(io_pair_list, func_list, times_list):
    assert len(io_pair_list) > 0
    solutions = afc(io_pair_list[0], func_list, times_list)
    return my_filter(lambda sol: my_all([satisfy(io_pair, sol) \
      for io_pair in io_pair_list], solutions)
```

2分

3分

漏写

```
[afc(io_pair, func_list, times_list)
    for io_pair in io_pair_list]
```


```
def gen_p():
  n = 2
  while True:
    if my_all([n % i != 0
               for i in range(2, n)]):
      yield n
    n += 1
```

多思考!
开动你的PyBrain

```
def extended_afc(io_pair_list, func_list, times_list):
    assert len(io_pair_list) > 0
    solutions = afc(io_pair_list[0], func_list, times_list)
    return my_filter(lambda sol: my_all([satisfy(io_pair, sol) \
      for io_pair in io_pair_list], solutions)
```

Think twice, code once.

祝同学们在下半学期的学习中取得更好的成绩!