NJU SICP

期中试卷讲解

jjppp

2024-11-15

1. 我负责...

- Environment Diagram
- Trees
- Remember Me!

2. Environment Diagram

2.1 命题意图

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

考点:

- name lookup
- function as values
- 故意有点难（没想到这么惨）

Cool Stuff:

2.1 命题意图

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

考点:

- name lookup
- function as values
- 故意有点难（没想到这么惨）

Cool Stuff:

- `fact` **有错**
  - 递归漏终止条件
  - `RecursionError`

2.1 命题意图

```
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

考点:

- name lookup
- function as values
- 故意有点难（没想到这么惨）

Cool Stuff:

- `fact` **有错**
  - 递归漏终止条件
  - `RecursionError`
- `fix` **修复**了有错的递归函数
  - 补上终止条件

2.1 命题意图

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

考点:

- name lookup
- function as values
- 故意有点难（没想到这么惨）

Cool Stuff:

- `fact` **有错**
  - 递归漏终止条件
  - `RecursionError`
- `fix` **修复**了有错的递归函数
  - 补上终止条件
- 青春版**动态更新**
  - 停服维护 vs 不停服维护

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Easy:

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Easy:

- fact(n): 阶乘函数
  - result = fact(2) = 2

2.2 观察

```
1 def fix(f):
2   def inner(n):
3     return 1 if n <= 1 else f(n)
4   return inner
5
6 fact = lambda n: n * fact(n - 1)
7 fact = fix(fact)
8 result = fact(2)
```

Easy:

- fact(n)：阶乘函数
  - result = fact(2) = 2
- 总共出现了三个函数
  1. fix
  2. inner
  3. lambda<6>

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Easy:

- `fact(n)`：阶乘函数
  - ▸ `result = fact(2) = 2`
- 总共出现了三个函数
  1. `fix`
  2. `inner`
  3. `lambda<6>`
- `fact` 被赋值了两次
  - ▸ `fact` 有两条边：`lambda<6>`，`inner`

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n <= 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Medium:

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Medium:

- parent frame:

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n <= 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Medium:

- parent frame:
  - parent[`fix`] = `global`

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n <= 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Medium:

- parent frame:
  - parent[`fix`] = `global`
  - parent[`inner`] = `fix`

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Medium:

- parent frame:
  - parent[`fix`] = `global`
  - parent[`inner`] = `fix`
  - parent[`lambda<6>`] = `global`

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Hard:

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Hard:

- 参数 `f`
  - 传入 `fact`，值为 `lambda<6>`

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Hard:

- 参数 `f`
  - 传入 `fact`，值为 `lambda<6>`
- 函数调用

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n ≤ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Hard:

- 参数 `f`
  - 传入 `fact`，值为 `lambda<6>`
- 函数调用
  - `fact = fix(fact)`
    - `global` ⇒ `fix`

2.2 观察

```python
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

Hard:

- 参数 `f`
  - 传入 `fact`，值为 `lambda<6>`
- 函数调用
  - `fact = fix(fact)`
    - `global` ⇒ `fix`
  - `result = fact(2)`
    - `global` ⇒ `fact{inner}(2)`
      ⇒ `f{lambda<6>}(2)`
      ⇒ `fact{inner}(1)`

3. Trees

3.1 命题意图

```python
def fix(f):
  def inner(n):
    return 1 if n ≤ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

考点:

- data abstraction
- recursion
- 送分

Cool Stuff:

3.1 命题意图

```
def fix(f):
  def inner(n):
    return 1 if n ⩽ 1 else f(n)
  return inner

fact = lambda n: n * fact(n - 1)
fact = fix(fact)
result = fact(2)
```

考点:

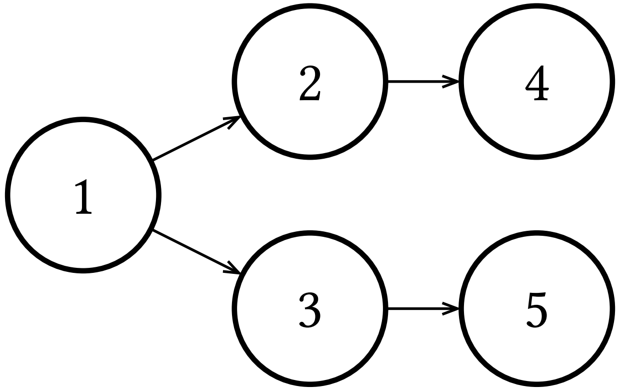
- data abstraction
- recursion
- 送分

Cool Stuff:

- “还差一个题，送点分吧”
- “上了实验课的同学应该都送到了”

3.2 `t1`

```
t1 = tree(1, [
                 tree(2, [
                           tree(4)]),
                 tree(3, [
                           tree(5)])])
```



改卷反馈：

- 大部分人都送到了
- 错误示范❌：
  - `[1, [2, [4]], [3, [5]]]`
  - `tree(1, [tree(2, tree(4))], [tree(3, tree(5))])`

3.3 map_tree

```
def map_tree(t, f):
  return tree(
    f(label(t)),
    [map_tree(b, f) for b in branches(t)]
  )
```

递归:

3.3 map_tree

```python
def map_tree(t, f):
  return tree(
    f(label(t)),
    [map_tree(b, f) for b in branches(t)]
  )
```

递归:

- `is_leaf`:
  直接调用 `f`
  作用在 `label(t)` 上

3.3 map_tree

```
def map_tree(t, f):
  return tree(
    f(label(t)),
    [map_tree(b, f) for b in branches(t)]
  )
```

递归:

- is_leaf:
  直接调用 f
  作用在 label(t) 上
- 否则枚举 branches
  递归地调用 map_tree

3.3 map_tree

```python
def map_tree(t, f):
  return tree(
    f(label(t)),
    [map_tree(b, f) for b in branches(t)]
  )
```

递归:

- is_leaf:
  直接调用 f
  作用在 label(t) 上
- 否则枚举 branches
  递归地调用 map_tree
- doctest 透露了上一题的答案

3.3 map_tree

```
def map_tree(t, f):
  return tree(
    f(label(t)),
    [map_tree(b, f) for b in branches(t)]
  )
```

递归:

- is_leaf:
  直接调用 f
  作用在 label(t) 上
- 否则枚举 branches
  递归地调用 map_tree
- doctest 透露了上一题的答案
- 答案不唯一
  中括号不是必须的(why?)
  不考虑 is_leaf 也 OK(why?)

3.3 map_tree

```python
def map_tree(t, f):
  return tree(
    f(label(t)),
    [map_tree(b, f) for b in branches(t)]
  )
```

改卷反馈:

3.3 `map_tree`

```
def map_tree(t, f):
  return tree(
    f(label(t)),
    [map_tree(b, f) for b in branches(t)]
  )
```

改卷反馈:

- `f(t)`❌:
  `f` 是作用在 `label` 上的函数
  学会读 doctest

3.4 height

```
def height(t, f):
  if is_leaf(t):
    return 1
  return 1 + max([
    height(b) for b in branches(t)
  ])
```

递归:

3.4 height

```
def height(t, f):
  if is_leaf(t):
    return 1
  return 1 + max([
    height(b) for b in branches(t)
  ])
```

递归:

- is_leaf: 返回 1

3.4 height

```
def height(t, f):
  if is_leaf(t):
    return 1
  return 1 + max([
    height(b) for b in branches(t)
  ])
```

递归:

- `is_leaf`: 返回 1
- 否则枚举 `branches` 递归地计算 `height`
- 然后返回最大值+1

3.4 height

```
def height(t, f):
  if is_leaf(t):
    return 1
  return 1 + max([
    height(b) for b in branches(t)
  ])
```

递归:

- `is_leaf`: 返回 1
- 否则枚举 `branches` 递归地计算 `height`
- 然后返回最大值+1
- 答案不唯一
  - `1+` 放里面
  - `sorted( ... )[-1]` 很有想象力
  - 中括号不是必须的(why?)

3.4 height

```
def height(t, f):
  if is_leaf(t):
    return 1
  return 1 + max([
    height(b) for b in branches(t)
  ])
```

改卷反馈:

3.4 height

```
def height(t, f):
  if is_leaf(t):
    return 1
  return 1 + max([
    height(b) for b in branches(t)
  ])
```

改卷反馈:

- 忘了+1
  学会读题

3.4 height

```
def height(t, f):
  if is_leaf(t):
    return 1
  return 1 + max([
    height(b) for b in branches(t)
  ])
```

改卷反馈:

- 忘了+1
  学会读题
- `sum()`
  学会读题

3.5 What went wrong?

改错:

Line #	Correct Answer	Comment
3	`is_leaft(t1)`	tree $\neq$ list
4	`tree(l1 + l2, [copy_tree(b) for b in branches(t2)]`	list $\neq$ tree
6	`tree(l1 + l2, [copy_tree(b) for b in branches(t1)]`	
10	`range(max(len(br1), len(br2)))`	
15	`copy_tree(br1[i])`	
18	`tree(l1 + l2, new_branches)`	

3.5 What went wrong?

改错:

Line #	Correct Answer	Comment
3	`is_leaft(t1)`	tree $\neq$ list
4	`tree(l1 + l2, [copy_tree(b) for b in branches(t2)]`	list $\neq$ tree
6	`tree(l1 + l2, [copy_tree(b) for b in branches(t1)]`	
10	`range(max(len(br1), len(br2)))`	
15	`copy_tree(br1[i])`	
18	`tree(l1 + l2, new_branches)`	

命题意图:

3.5 What went wrong?

改错:

Line #	Correct Answer	Comment
3	`is_leaft(t1)`	tree $\neq$ list
4	`tree(l1 + l2, [copy_tree(b) for b in branches(t2)]`	list $\neq$ tree
6	`tree(l1 + l2, [copy_tree(b) for b in branches(t1)]`	
10	`range(max(len(br1), len(br2)))`	
15	`copy_tree(br1[i])`	
18	`tree(l1 + l2, new_branches)`	

命题意图:

- 题中所有错误均来自 OJ 真实提交

3.5 What went wrong?

改错:

Line #	Correct Answer	Comment
3	`is_leaft(t1)`	tree $\neq$ list
4	`tree(l1 + l2, [copy_tree(b) for b in branches(t2)]`	list $\neq$ tree
6	`tree(l1 + l2, [copy_tree(b) for b in branches(t1)]`	
10	`range(max(len(br1), len(br2)))`	
15	`copy_tree(br1[i])`	
18	`tree(l1 + l2, new_branches)`	

命题意图:

- 题中所有错误均来自 OJ 真实提交
- 青春版助教体验
  “请问我哪里错了？”

3.5 What went wrong?

改错:

Line #	Correct Answer	Comment
3	`is_leaft(t1)`	tree $\neq$ list
4	`tree(l1 + l2, [copy_tree(b) for b in branches(t2)]`	list $\neq$ tree
6	`tree(l1 + l2, [copy_tree(b) for b in branches(t1)]`	
10	`range(max(len(br1), len(br2)))`	
15	`copy_tree(br1[i])`	
18	`tree(l1 + l2, new_branches)`	

命题意图:

- 题中所有错误均来自 OJ 真实提交
- 青春版助教体验
  “请问我哪里错了？”
- “加点空行，坑一下同学？”

3.5 What went wrong?

改错:

Line #	Correct Answer	Comment
3	`is_leaft(t1)`	tree $\neq$ list
4	`tree(l1 + l2, [copy_tree(b) for b in branches(t2)]`	list $\neq$ tree
6	`tree(l1 + l2, [copy_tree(b) for b in branches(t1)]`	
10	`range(max(len(br1), len(br2)))`	
15	`copy_tree(br1[i])`	
18	`tree(l1 + l2, new_branches)`	

命题意图:

- 题中所有错误均来自 OJ 真实提交
- 青春版助教体验
  “请问我哪里错了？”
- “加点空行，坑一下同学？”

改卷反馈:

3.5 What went wrong?

改错:

Line #	Correct Answer	Comment
3	`is_leaft(t1)`	tree $\neq$ list
4	`tree(l1 + l2, [copy_tree(b) for b in branches(t2)]`	list $\neq$ tree
6	`tree(l1 + l2, [copy_tree(b) for b in branches(t1)]`	
10	`range(max(len(br1), len(br2)))`	
15	`copy_tree(br1[i])`	
18	`tree(l1 + l2, new_branches)`	

命题意图:

- 题中所有错误均来自 OJ 真实提交
- 青春版助教体验
  “请问我哪里错了？”
- “加点空行，坑一下同学？”

改卷反馈:

- 4 和 6 对照着看，6 分

3.5 What went wrong?

改错:

Line #	Correct Answer	Comment
3	`is_leaft(t1)`	tree $\neq$ list
4	`tree(l1 + l2, [copy_tree(b) for b in branches(t2)]`	list $\neq$ tree
6	`tree(l1 + l2, [copy_tree(b) for b in branches(t1)]`	
10	`range(max(len(br1), len(br2)))`	
15	`copy_tree(br1[i])`	
18	`tree(l1 + l2, new_branches)`	

命题意图:

- 题中所有错误均来自 OJ 真实提交
- 青春版助教体验
  “请问我哪里错了？”
- “加点空行，坑一下同学？”

改卷反馈:

- 4 和 6 对照着看，6 分
- 漏了 10

3.5 What went wrong?

改错:

Line #	Correct Answer	Comment
3	`is_leaft(t1)`	tree $\neq$ list
4	`tree(l1 + l2, [copy_tree(b) for b in branches(t2)]`	list $\neq$ tree
6	`tree(l1 + l2, [copy_tree(b) for b in branches(t1)]`	
10	`range(max(len(br1), len(br2)))`	
15	`copy_tree(br1[i])`	
18	`tree(l1 + l2, new_branches)`	

命题意图:

- 题中所有错误均来自 OJ 真实提交
- 青春版助教体验
  “请问我哪里错了？”
- “加点空行，坑一下同学？”

改卷反馈:

- 4 和 6 对照着看，6 分
- 漏了 10
- 不 `copy_tree` 的也给了分

3.5 What went wrong?

改错:

Line #	Correct Answer	Comment
3	`is_leaft(t1)`	tree $\neq$ list
4	`tree(l1 + l2, [copy_tree(b) for b in branches(t2)]`	list $\neq$ tree
6	`tree(l1 + l2, [copy_tree(b) for b in branches(t1)]`	
10	`range(max(len(br1), len(br2)))`	
15	`copy_tree(br1[i])`	
18	`tree(l1 + l2, new_branches)`	

命题意图:

- 题中所有错误均来自 OJ 真实提交
- 青春版助教体验
  “请问我哪里错了？”
- “加点空行，坑一下同学？”

改卷反馈:

- 4 和 6 对照着看，6 分
- 漏了 10
- 不 `copy_tree` 的也给了分
- 好多同学空着(why?)

4. Remember Me!

4.1 命题意图

考点:

- 长文阅读(~900 词)
- higher order functions
- pure functions
- ADT
- 在课程 code 中出现过
  (同学的提问也能学到东西)

Cool Stuff:

4.1 命题意图

考点:

- 长文阅读(~900 词)
- higher order functions
- pure functions
- ADT
- 在课程 code 中出现过
  (同学的提问也能学到东西)

Cool Stuff:

- Python @decorator 的原理
  有没有研究过 `cats.py` 的代码?
- （几乎）全自动的程序优化

4.2 count_calls

```python
def count_calls(n):
  if n == 1 or n == 2:
    return 1
  return count_calls(n - 1) + count_calls(n - 2) + 1
```

思路：

4.2 count_calls

```python
def count_calls(n):
  if n == 1 or n == 2:
    return 1
  return count_calls(n - 1) + count_calls(n - 2) + 1
```

思路:

- 数 call tree 节点数

4.2 count_calls

```python
def count_calls(n):
  if n == 1 or n == 2:
    return 1
  return count_calls(n - 1) + count_calls(n - 2) + 1
```

思路:

- 数 call tree 节点数
- `count_calls` 的增长速度:

4.2 count_calls

```python
def count_calls(n):
  if n == 1 or n == 2:
    return 1
  return count_calls(n - 1) + count_calls(n - 2) + 1
```

思路:

- 数 call tree 节点数
- `count_calls` 的增长速度:
  1. 注意到 `count_calls(n)` $\geq$ `fib(n)`
  2.
  3.

4.2 count_calls

```
def count_calls(n):
  if n == 1 or n == 2:
    return 1
  return count_calls(n - 1) + count_calls(n - 2) + 1
```

思路:

- 数 call tree 节点数
- count_calls 的增长速度:
  1. 注意到 count_calls(n) $\geq$ fib(n)
  2. 且 fib(n) 指数级增长
  3.

4.2 count_calls

```python
def count_calls(n):
  if n == 1 or n == 2:
    return 1
  return count_calls(n - 1) + count_calls(n - 2) + 1
```

思路:

- 数 call tree 节点数
- count_calls 的增长速度:
  1. 注意到 count_calls(n) $\geq$ fib(n)
  2. 且 fib(n) 指数级增长
  3. 在黑板上写一下

改卷反馈:

- 不少人写了又一个 fib
  题目提示 count_calls(5)是 9
  学会读题

4.2 count_calls

```python
def count_calls(n):
  if n == 1 or n == 2:
    return 1
  return count_calls(n - 1) + count_calls(n - 2) + 1
```

思路:

- 数 call tree 节点数
- `count_calls` 的增长速度:
  1. 注意到 `count_calls(n)` $\geq$ `fib(n)`
  2. 且 `fib(n)` 指数级增长
  3. 在黑板上写一下

改卷反馈:

- 不少人写了又一个 `fib`
  题目提示 `count_calls(5)`是 9
  学会读题
- 有同学试图写通项
  很可惜写错了
  (通项和 `fib` 类似)

4.3 Pure

```python
import random
def lottery():
  return random.randint(6, 71)
```

F

```python
def fact(n):
  return 1 if n == 1 else fact(n - 1) * n
```

T

```python
t = []
def append_len(x):
  t.append(x)
  return len(t)
```

F

```python
print(append_len('A'))
print(append_len('B'))
```

```python
def fun(x):
  def safe(y, total):
    nonlocal x
    if y <= 0:
      return total
    x = (y - 1) * 2
    return safe(y - 1, x * (total + 1))
  return safe(x, 0)
```

T

nonlocal x 是酱油
调用 safe 使 y 少 1
当 y ⩽ 0, total 为 0
(有限度的)副作用

4.4 Rewrite Me?

```python
mem = memory()
def fib_r(n):
  if recall(mem, n):
    return recall(mem, n)
  Fn = 1
  if n >= 3:
    Fn = fib_r(n - 1) + fib_r(n - 2)
  remember(mem, n, Fn)
  return Fn
```

改卷反馈:

4.4 Rewrite Me?

```
mem = memory()
def fib_r(n):
  if recall(mem, n):
    return recall(mem, n)
  Fn = 1
  if n >= 3:
    Fn = fib_r(n - 1) + fib_r(n - 2)
  remember(mem, n, Fn)
  return Fn
```

改卷反馈:

- 不少同学第二个空填 `not recall`

4.4 Rewrite Me?

```
mem = memory()
def fib_r(n):
  if recall(mem, n):
    return recall(mem, n)
  Fn = 1
  if n >= 3:
    Fn = fib_r(n - 1) + fib_r(n - 2)
  remember(mem, n, Fn)
  return Fn
```

改卷反馈:

- 不少同学第二个空填 `not recall`
  - 会无限递归，直到 RecursionError
    `fib_r(3) ⇒ fib_r(2) ⇒ fib_r(1) ⇒ fib_r(0) ⇒ …`
- 有同学填了 `n >= 2`

4.4 Rewrite Me?

```
mem = memory()
def fib_r(n):
  if recall(mem, n):
    return recall(mem, n)
  Fn = 1
  if n >= 3:
    Fn = fib_r(n - 1) + fib_r(n - 2)
  remember(mem, n, Fn)
  return Fn
```

改卷反馈:

- 不少同学第二个空填 `not recall`
  - 会无限递归，直到 RecursionError
    `fib_r(3) ⇒ fib_r(2) ⇒ fib_r(1) ⇒ fib_r(0) ⇒ ...`
- 有同学填了 `n >= 2`
  - 很可惜错了(why?)

4.5 Remember Me!

```
def remember_me(f):
  mem = memory()
  def inner(n):
    if recall(mem, n):
      return recall(mem, n)
    Fn = f(n)
    remember(mem, n, Fn)
    return Fn
  return inner
```

改卷反馈:

4.5 Remember Me!

```
def remember_me(f):
  mem = memory()
  def inner(n):
    if recall(mem, n):
      return recall(mem, n)
    Fn = f(n)
    remember(mem, n, Fn)
    return Fn
  return inner
```

改卷反馈:

- 一些同学认为这里还在算 `fib`
  - `remember_me` 是通用的记忆化函数
    我们在后面还记忆化了 `count_k`

4.5 Remember Me!

```
def remember_me(f):
  mem = memory()
  def inner(n):
    if recall(mem, n):
      return recall(mem, n)
    Fn = f(n)
    remember(mem, n, Fn)
    return Fn
  return inner
```

改卷反馈:

- 一些同学认为这里还在算 `fib`
  - `remember_me` 是通用的记忆化函数
    我们在后面还记忆化了 `count_k`
- 答案不唯一

4.5 Remember Me!

```
def two_to_one(f):
    return lambda nk: f(nk[0], nk[1])

count_k = two_to_one(count_k)

# old_count_k: (n: int, k: int) → int
# new_count_k: (nk: (int, int)) → int
```

4.5 Remember Me!

```
def two_to_one(f):
    return lambda nk: f(nk[0], nk[1])

count_k = two_to_one(count_k)

# old_count_k: (n: int, k: int) → int
# new_count_k: (nk: (int, int)) → int

new_count_k((5, 3))
  ⇒ (lambda nk: old_count_k(nk[0], nk[1]))((5, 3))
  ⇒ old_count_k(5, 3)
  ⇒ count_k(5 - i, 3) # Which count_k?
                      # new_count_k!
                      # TypeError
```

改卷反馈:

- 我是故意的(1 分)
- 灵感源自群里关于 argument unpacking 的提问
- 不少人算了半天 13/其它数
- 做对的同学不多

5. Bonus

改卷反馈:

- 得分率不高
  - 奖励认真读教材的同学
  - 也可能是没做到最后一题?
- 单词写不对
  - Interptation
  - ... of Computer Science
  - ... of Composing Programs
  - 英语很重要