NJU SICP

Tutorial

jjppp

2024-11-01

1. 期中考试

- 考试时间: 2024-11-13
- 试卷讲解: 2024-11-15

1.1 题型

1. WWPD

code	output
`print(print(114), 514)`	?

2. Environment Diagram

```
n = 24
def man(n, f):
  return f(2 * n)
def out(m):
  return n + m

man(8)(out)
```

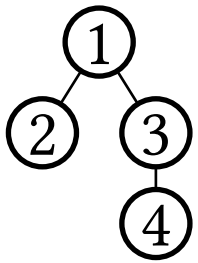
1.1 题型

3. Code Completion

```
def my_reverse(lst):
  return ____
```

4. Drawings

   draw `tree(1, [tree(2), tree(3, [tree(4)])])`



5. Others

2. Data Abstraction

2.1 Review

Data Abstraction:

- **Details** do not matter
- Never make **assumptions**

2.1 Review

Data Abstraction:

- **Details** do not matter
- Never make **assumptions**

2.1 Review

Example1:

Data Abstraction:

- **Details** do not matter
- Never make **assumptions**

Data Abstraction:

- **Details** do not matter
- Never make **assumptions**

Example1: Courses

With a course, you can

- **take** it
- **drop** it
- earn **credits**
- ...

You cannot

- **assume** it's easy (SICP)
- **presume** it takes exams (悦读)
- ...

Example2:

Data Abstraction:

- **Details** do not matter
- Never make **assumptions**

Data Abstraction:

- **Details** do not matter
- Never make **assumptions**

Example2: Labs

With a submitted answer, TA's can

- **run** it
- **score** it
- ...

TA's cannot (and do not care to)

- **assume** its algorithm (recursive, iterative, ...)
- **presume** variable names
- ...

2.1 Review

Example3:

Data Abstraction:

- **Details** do not matter
- Never make **assumptions**

Data Abstraction:

- **Details** do not matter
- Never make **assumptions**

Example3: Python

With python interpreter, one can

- **run** programs
- **debug** programs
- ...

One should (probably) not

- **care** about computer brands (dell, lenoveo, huawei, ...)
- **assume** who's running it
- ...

2.2 Pair Abstraction

```
def fst(p):
    return p(0)

def snd(p):
    return p(1)
```

1.
2.
3.

2.2 Pair Abstraction

```
def fst(p):
    return p(0)

def snd(p):
    return p(1)
```

1. p is function
2.
3.

2.2 Pair Abstraction

```
def fst(p):
    return p(0)

def snd(p):
    return p(1)
```

1. p is function
2. `pair(x, y)(0) == x`
3.

2.2 Pair Abstraction

```
def fst(p):
    return p(0)

def snd(p):
    return p(1)
```

1. p is function
2. `pair(x, y)(0) == x`
3. `pair(x, y)(1) == y`

2.2 Pair Abstraction

```
def fst(p):
    return p(0)

def snd(p):
    return p(1)
```

1. p is function
2. `pair(x, y)(0) == x`
3. `pair(x, y)(1) == y`

```
def pair(x, y):
  def inner(i):
    return y if i else x
  return inner
```

2.2 Pair Abstraction

```
def fst(p):
    return p(0)

def snd(p):
    return p(1)
```

1. p is function
2. pair(x, y)(0) == x
3. pair(x, y)(1) == y

```
def pair(x, y):
  def inner(i):
    return y if i else x
  return inner
```

Important:

- pairs **can be** functions
- pairs are not **necessarily** functions

2.3 Pair Abstraction

```
def change_fst(p, v):
    return pair(v, snd(p))
```

2.3 Pair Abstraction

```
def change_fst(p, v):
    return pair(v, snd(p))
```

1. Only `pair()` and `snd()` used
2. Don't know what `p` actually is
3. Works for any `pair`

2.3 Pair Abstraction

```
def change_fst(p, v):
    return pair(v, snd(p))
```

1. Only `pair()` and `snd()` used
2. Don't know what `p` actually is
3. Works for any `pair`

Important:

- `pairs` **can be** functions
- `pairs` are not **always** functions

3. Lists

3.1 List Comprehension

Deep

```
[x[:] for x in l]
```

Reverse

```
l[::-1]
```

Split

```
([x for x in l if f(x)], [x for x in l if not f(x)])
```

Couple

```
[[lst1[i], lst2[i]] for i in range(len(lst1))]
```

4. Trees

- **tree $\neq$ list**

4.1 Add Trees

4.1 Add Trees

- `label(t)` may **not** be a number

  ```
  label(t1) + label(t2) != label(t2) + label(t1)
  label(t1) + 0 != label(t1)
  ```

4.1 Add Trees

- `label(t)` may **not** be a number

  `label(t1) + label(t2) != label(t2) + label(t1)`

  `label(t1) + 0 != label(t1)`

- returns a new **tree**

  `[0]`, `[label(t1) + label(t2), branches]`, ...

4.1 Add Trees

- `label(t)` may **not** be a number

  `label(t1) + label(t2) != label(t2) + label(t1)`

  `label(t1) + 0 != label(t1)`

- returns a new **tree**

  `[0]`, `[label(t1) + label(t2), branches]`, ...

- 

4.1 Add Trees

Idea:

`add_tree(t1, t2)` returns a new tree `tree(new_label, new_branches)`

1.
2.

4.1 Add Trees

Idea:

`add_tree(t1, t2)` returns a new tree `tree(new_label, new_branches)`

1. new label: `label(t1) + label(t2)`
2.

4.1 Add Trees

Idea:

`add_tree(t1, t2)` returns a new tree `tree(new_label, new_branches)`

1. new label: `label(t1) + label(t2)`
2. new branches:
    1.
    2.

4.1 Add Trees

Idea:

`add_tree(t1, t2)` returns a new tree `tree(new_label, new_branches)`

1. new label: `label(t1) + label(t2)`
2. new branches:
    1. first `min(len(branches(t1)), len(branches(t2)))` branches
       call `add_tree` recursively
    2.

4.1 Add Trees

Idea:

`add_tree(t1, t2)` returns a new tree `tree(new_label, new_branches)`

1. new label: `label(t1) + label(t2)`
2. new branches:
    1. first `min(len(branches(t1)), len(branches(t2)))` branches
       call `add_tree` recursively
    2. other branches
       call `copy_tree`

4.1 Add Trees

Bonus

4.1 Add Trees

Bonus

Try implement `tree(label, branches=[])` and the corresponding selectors using

4.1 Add Trees

Bonus

Try implement `tree(label, branches=[])` and the corresponding selectors using

- `(label, branches)`
- `[branches, label]`
- `pair(label, branches)`
- functions

4.1 Add Trees

Bonus

Try implement `tree(label, branches=[])` and the corresponding selectors using

- `(label, branches)`
- `[branches, label]`
- `pair(label, branches)`
- functions

and see if your solution still works.

4.2 Big Path

Idea: recursion

4.2 Big Path

Idea: recursion

1. Base case:
    1. `label(t)<n`: `0` paths
    2. `is_leaf(t)`: `1` path
2.

4.2 Big Path

Idea: recursion

1. Base case:
    1. `label(t)<n`: `0` paths
    2. `is_leaf(t)`: `1` path
2. Inductive case:

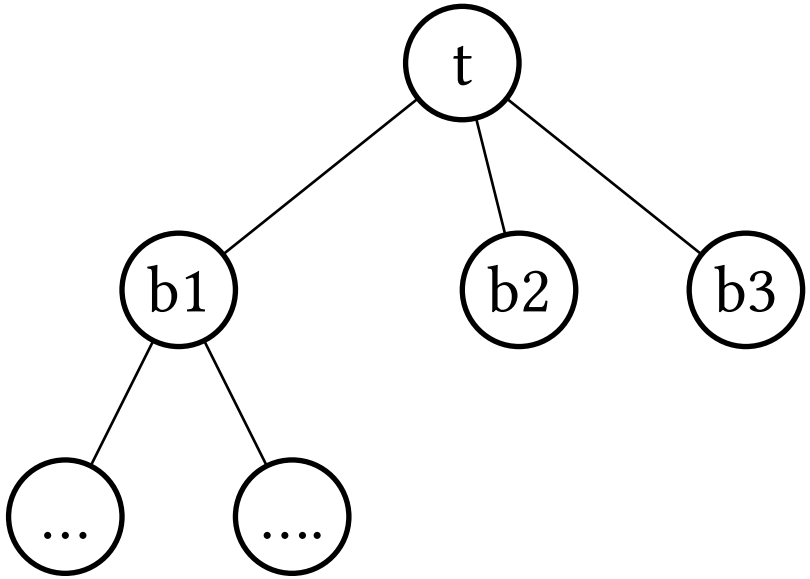
```
for b in branches(t):
  s += bigpath(b, n-label(t))
```

4.2 Big Path

Idea: recursion

1. Base case:
    1. `label(t)<n`: `0` paths
    2. `is_leaf(t)`: `1` path
2. Inductive case:

```
for b in branches(t):
  s += bigpath(b, n-label(t))
```



5. Take-aways

- **期中考试**
- **Abstraction**