CS1010S Tutorial 5

Sean Ng

AY2018/19 Sem 1, Week 8

Updated 2019-03-12 at 15:03:52

Materials: pengnam.github.io/1010s

Contact: seanngpengnam@u.nus.edu

Feedback

Tuples in ADTs

- Tuples have been very common in ADTs so far
- But they can become very confusing when ADTs have ADTs in ADTs have ADTs...
- Deeply nested ADTs can mean deeply nested tuples
- Which can be difficult to visualise

Tuples in ADTs

An mild example from the train mission: (Try to identify this ADT)

```
(('TRAIN 0-0',), (True, ('CC4', 'Promenade'),
 ('CC3', 'Esplanade')), datetime.datetime(2016, 1, 1, 9, 27))
```

Some tips to visualising nested tuples in ADTs:

1. Good formatting (given a `print` output)
2. Following abstraction (given a variable)

Tuples in ADTs

Good formatting:

```
(
    (
        'TRAIN 0-0',
    ),
    (
        True,
        ('CC4', 'Promenade'),
        ('CC3', 'Esplanade')
    ),
    datetime.datetime(2016, 1, 1, 9, 27)
)
```

The *Train* ADT, *TrainPosition* ADT, and *time* align themself.

Tuples in ADTs

Following abstraction:

In your mind, don't start thinking like this:

```
(
    (
        'TRAIN 0-0',
    ),
    (
        True,
        ('CC4', 'Promenade'),
        ('CC3', 'Esplanade')
    ),
    datetime.datetime(2016, 1, 1, 9, 27)
)
```

Rather, follow the abstractions:

```
(train, train_position, time)
```

Tuples in ADTs

Then, unpack them as you go.

1. To find the name of the next station:
   - `(train, train_position, time)`
2. Note that the next *Station* is part of the *TrainPosition*...
   - `(train, (is_moving, from_station, to_station), time)`
3. Note that the name is part of the next *Station*...
   - `(train, (is_moving, (station_code, station_name), to_station), time)`

Tuples in ADTs

Then, unpack them as you go.

1. To find the name of the next station:
   - `(train, train_position, time)`
2. Note that the next *Station* is part of the *TrainPosition*...
   - `(train, (is_moving, from_station, to_station), time)`
3. Note that the name is part of the next *Station*...
   - `(train, (is_moving, (station_code, station_name), to_station), time)`

Tuples in ADTs

Then, unpack them as you go.

1. To find the name of the next station:
   - `(train, train_position, time)`
2. Note that the next *Station* is part of the *TrainPosition*...
   - `(train, (is_moving, from_station, to_station), time)`
3. Note that the name is part of the next *Station*...
   - `(train, (is_moving, (station_code, station_name), to_station), time)`

Tuple Concatenation

- Tuple concatenation (with the +) combines two *tuples*
- Tuples are immutable, so a new tuple is created
- Be careful about nested tuples!

Tuple Concatenation

- Tuple concatenation (with the +) combines two *tuples*
- Tuples are immutable, so a new tuple is created
- Be careful about nested tuples!

Tuple Concatenation

- Tuple concatenation (with the +) combines two *tuples*
- Tuples are immutable, so a new tuple is created
- Be careful about nested tuples!

Tuple Concatenation

Visualising tuple cocatenation:

1. In order to add the two tuples:

   ```
   - (1, (2,), 3) + (4,)
   ```

2. Replace the middle `) + (` with a `,` (comma)

   ```
   - (1, (2,), 3, 4,)
   ```

3. Remove the trailing comma

   ```
   - (1, (2,), 3, 4)
   ```

Tuple Concatenation

Visualising tuple cocatenation:

1. In order to add the two tuples:

   ```
   - (1, (2,), 3) + (4,)
   ```

2. Replace the middle `) + (` with a `,` (comma)

   ```
   - (1, (2,), 3, 4,)
   ```

3. Remove the trailing comma

   ```
   - (1, (2,), 3, 4)
   ```

Tuple Concatenation

Visualising tuple cocatenation:

1. In order to add the two tuples:

   - `(1, (2,), 3) + (4,)`

2. Replace the middle `) + (` with a `,` (comma)

   - `(1, (2,), 3, 4,)`

3. Remove the trailing comma

   - `(1, (2,), 3, 4)`

Tuple Concatenation

Visualising tuple cocatenation:

1. In order to add the two tuples:

   ```
   - (1, (2,), 3) + (4,)
   ```

2. Replace the middle `) + (` with a `,` (comma)

   ```
   - (1, (2,), 3, 4,)
   ```

3. Remove the trailing comma

   ```
   - (1, (2,), 3, 4)
   ```

Tuple Concatenation

- Caveat: the following doesn't work (why?)
  - `(1, 2, 3) + ((1,))`

Tuple Index vs Slicing

- Note that there are two different behaviours for attempting to 'assess' outside of a tuple's index.
    1. Indexing—throws an `IndexError`
    2. Slicing—returns an empty tuple

Tuple Index vs Slicing

- Note that there are two different behaviours for attempting to 'assess' outside of a tuple's index.
  1. Indexing—throws an `IndexError`
  2. Slicing—returns an empty tuple

Tuple Index vs Slicing

- Note that there are two different behaviours for attempting to 'assess' outside of a tuple's index.
  1. Indexing—throws an `IndexError`
  2. Slicing—returns an empty tuple

Tuple Index vs Slicing

Indexing

```
a = (1, 2, 3)
a[3] # IndexError: tuple index out of range
```

Slicing

```
a = (1, 2, 3)
a[3:] # ()
```

Tuple Index vs Slicing

Indexing

```
a = (1, 2, 3)
a[3] # IndexError: tuple index out of range
```

Slicing

```
a = (1, 2, 3)
a[3:] # ()
```

Tutorial

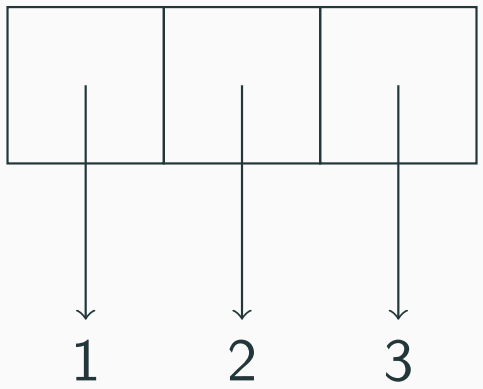
Box-and-pointers

Most of the time, it's easier to draw 'inside-out'

E.g. to draw `(((1, 2, 3), 4, 5), 6)`

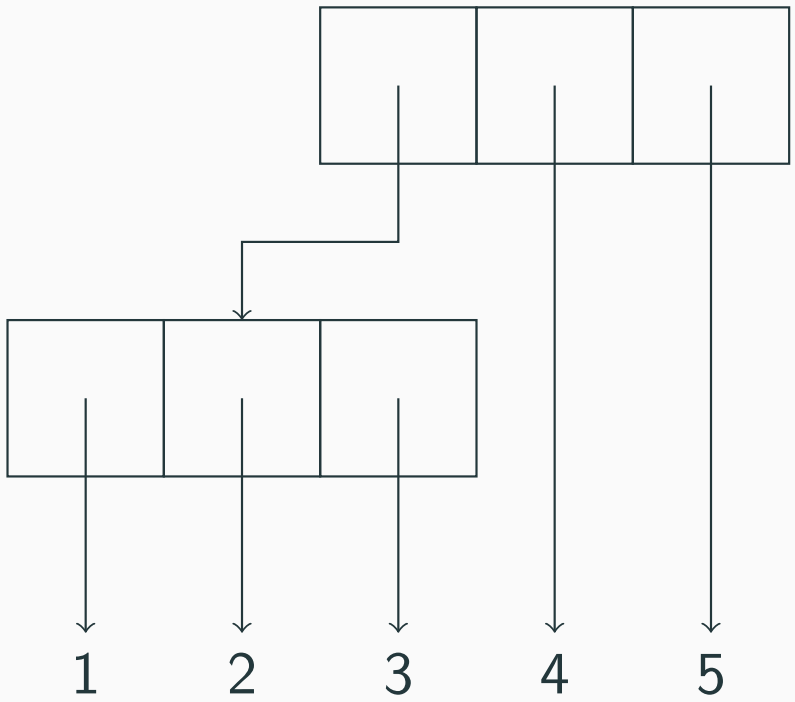
Box-and-pointers

```
((1, 2, 3), 4, 5), 6)
```



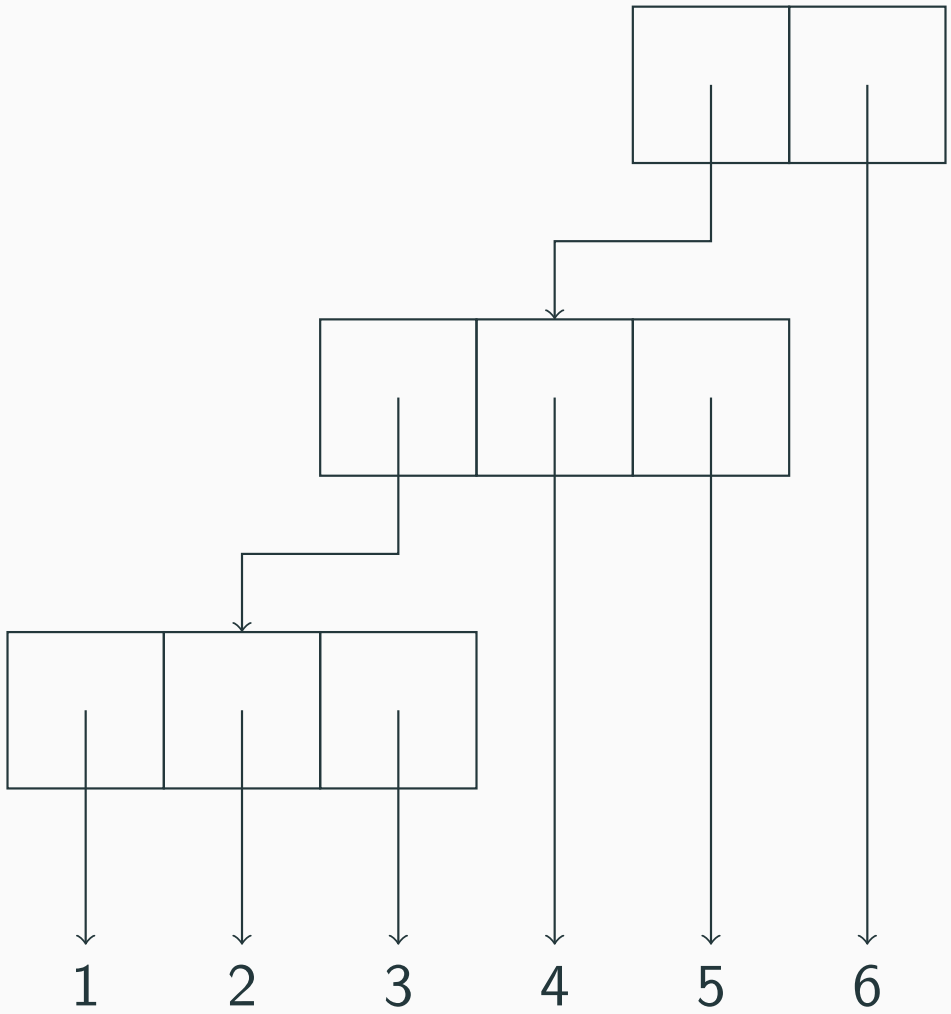
Box-and-pointers

```
((1, 2, 3), 4, 5), 6)
```



Box-and-pointers

```
(((1, 2, 3), 4, 5), 6)
```



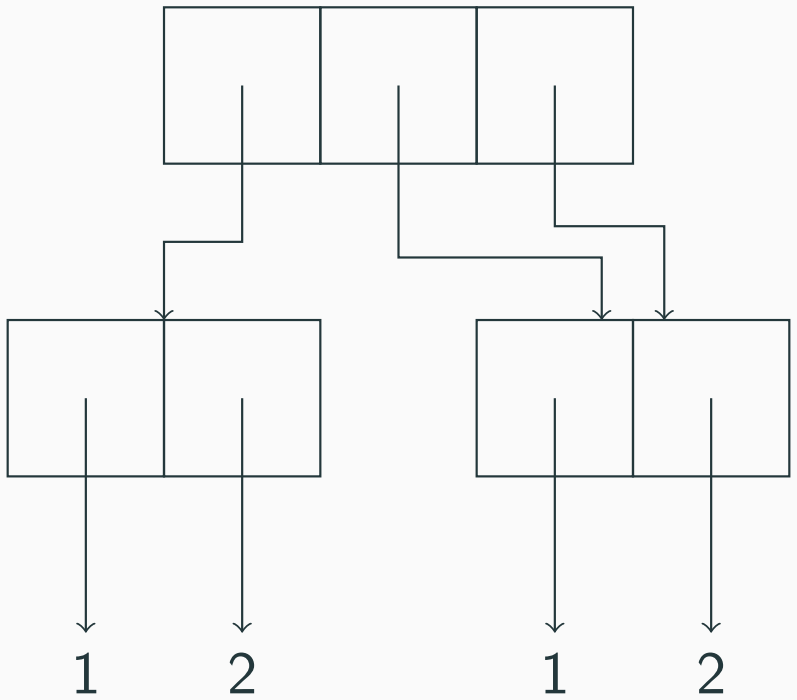
Box-and-pointers

Additional practice? repl.it/@ningyuansg/Tree-Generator

(Hit the green 'play' button at the top)

Box-and-pointers: Identity

```
a = (1, 2)
b = ((1, 2), a, a) # draw the box-and-pointer for b
```



Box-and-pointers: Identity

```
a = (1, 2)
b = ((1, 2), a, a)

b[0] is b[1] # False
b[1] is b[2] # True
b[0] is b[2] # False
```

Even Rank

Write a Python function called `even_rank` that takes in a tuple as its only argument and returns a tuple containing all the elements of even rank every second element from the left) from the input tuple

Odd Even Sum

Write a function called `odd_even_sums` that takes in a tuple of numbers as its only argument and returns a tuple of two elements: the first is the sum of all odd- ranked numbers in the input tuple, whereas the second element is the sum of all even-ranked elements in the input.

Towers of Hanoi

To move a stack of n from `src` to `dest`,

1. Move a stack of $n-1$ from `src` to `aux`
2. Move the n^{th} disk to `dest`
3. Move the stack of $n-1$ from `aux` to `dest`

Wishful thinking: steps 1 and 3.

Recall *Fibonacci*. To calculate the n^{th} Fibonacci number,

1. Calculate the $n-1^{\text{th}}$ term
2. Calculate the $n-2^{\text{th}}$ term
3. Add the $n-1^{\text{th}}$ and $n-2^{\text{th}}$ term

Wishful thinking: steps 1 and 2.

We resolve wishful thinking by solving the base case.

Towers of Hanoi