course: CSC 135

instructor: Ted Krovetz

related_notes: 2022-02-24

# Tail Recursion Optimization

W08.4 | Thursday, February 24, 2022 | 08:57 AM

## Agenda

1. Traversing HAMT
2. Tail recursion
3. Code Step-by-step

## Traversing HAMT - Logic

```
visit(self):
        # Do something here (before visiting children)
        for i = 0 to deg-1
                if child[i] not None:
                        visit(child[i])
        # Do something here (after visiting children)
```

## Tail recursion

- Recursion may break one's stack frame: O(n) or worse
- **Tail call optimization**: If the last operation is a recursive call, the recursion can be turned into a loop.

```python
def foo(x):
        if x is small:
                # solve directly
        else:
                # ...
                foo(smaller_x) # very last work done in branch of recursive function
```

The sample pseudo-code is effectively the same as the loop-based approach below

```
def foo(x):
        while True:
                if x is small:
                        # solve directly
                        return
                else:
                        # ...
                        x = smaller x
```

Java nor Python will optimize your code; however, GCC, a **C** compiler, will do it for you

## Greatest Common Divisor `gcd(x,y)` - Euclid's Algorithm

Pseudo-code recursion approach

```
def gcd(x,y):
        if y == 0:
                return x
        else:
                return gcd(y, x % y)
```

Pseudo-code loop-based approach

- Avoid a stack call overhead

```
def gcd(x,y):
        while True:
                if y == 0:
                        return x
                else:
                        (x,y) = (y, x & y)
```

## Factorials

Can't be turned into a loop

```
def fact(x):
        if x == 0:
                return 1
        else:
                return x \* fact(x-1)
```

You can get around this via an **accumulator** where it works out the solution as you go down, before the call, and NOT provide a result after the call

### Factorials - Accumulator Version

```
def fact(x, accu):
        if x == 0:
                return accu
        else:
                return fact(x-1, accu \* x)
```

### Factorials - Loop Version of Accumulator Version

```
def fact(x, accu = 1):
        while True:
                if x == 0:
                        return accu
                else:
                        (x, accu) = (x-1, x \* accu)
```

## Another Example - Rewrite the following function to use-tail recursion

```
# Multiplies a and non-negative b using repeated addition
def mult(a, b):
    if b == 0:
        return 0
    else:
        return a + mult(a, b-1)
```

### Tail Recursion

```
# Multiplies a and non-negative b using repeated addition
def mult(a, b, acc = 0):
    if b == 0:
        return acc
    else:
        return mult(a, b-1, a + acc)
```

## Cues/Questions

- O(n) stack frames problem for HAMT
    - Test cases professor uses won't blow-up the stack counter
    - Nevertheless, best to implement a loop-based solution