CS240

Nathan Sprague

October 17, 2014

Preamble: Summations

- What is the average case running time of sequential search?
- Average cost $= \frac{\text{Sum over cost of all possible cases}}{\#\ \text{possible cases}}$
- Assume a successful search, all locations equally likely...

Average Cost of Sequential Search

Useful summation: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Average Cost:

$$\frac{1+2+...+n}{n} =$$

$$\frac{\sum_{i=1}^{n} i}{n} =$$

$$\frac{n(n+1)/2}{n} = \frac{(n+1)}{2}$$

Another Useful Summation

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

Counting Activations With Recurrences

Draw a recursion trace for `split(3)`:

```
def split(n):
    if n == 0:
        return 1
    else:
        return split(n - 1) + split(n - 1)
```

Counting Activations With Recurrences

Draw a recursion trace for `split(3)`:

```python
def split(n):
    if n == 0:
        return 1
    else:
        return split(n - 1) + split(n - 1)
```

$F(0) = 1$
$F(n) = 2F(n-1) + 1$

Counting Operations

More typically, we want to count the number of steps taken by a recursive algorithm:

```
def split1(n):
    for i in range(n + 2):
        do_something() # We need to count this!
    if n == 0:
        return 1
    else:
        return split1(n - 1) + split1(n - 1)
```

Counting Operations

More typically, we want to count the number of steps taken by a recursive algorithm:

```
def split1(n):
    for i in range(n + 2):
        do_something() # We need to count this!
    if n == 0:
        return 1
    else:
        return split1(n - 1) + split1(n - 1)
```

$$F(0) = 2$$
$$F(n) = 2F(n-1) + n + 2$$

ANALYZING RECURSION: STAGE 1 - Develop a Recurrence

- Develop a recurrence relation that describes the number of times the basic operation occurs in the worst (best, average) case:
- Typically:

Initial Conditions

$$T(size\_of\_base\_case) = \#operations\_required\_for\_base\_case$$

Recurrence Relation

$$T(n) = \#recursive\_calls \times T(size\_of\_calls) + \#operations\_in\_call$$

Recursive Warm-Up

```
def fun3(n)
    if n == 0:
        return 20
    else:
        result = 0
        for i in range(4):
            result += 1
        return result + fun3(n - 1)
```

Let's develop an equation describing how many additions will be performed:

$T(0) = ??$

$T(n) = ??$

Recurrences

We can express this as a recurrence :

```
def fun3(n)
    if n == 0:
         return 20
    else:
        result = 0
        for i in range(4):
           result += 1
        return result + fun3(n-1)
```

Let's develop an equation describing how many additions will be performed:

$T(0) = 0$

$T(n) = 5 + T(n-1)$

Recurrence Exercise

Develop a recurrence that describes the number of additions:

```
def fun3(n)
    if n == 0:
        return 20
    else:
        result = 2 * n
        return result - (fun3(n-1) + fun3(n-1))
```

STAGE 2 - Solve the Recurrence

- One approach is the method of back substitution:
  1. Expand the recurrence by repeated substitution until a pattern emerges.
  2. Characterize the pattern by expressing the recurrence in terms of n and i, (where i is the number of substitutions).
  3. Substitute for i an expression that will remove the recursive term.
  4. Manipulate the result to achieve a closed-form solution.

STAGE 3 Check Your Answer

- Make sure that the recurrence and the closed form solution agree for several values of n.

Example: Recursive Binary Search, Stage 1

Worst-case recurrence:
$W(1) = 1$
$W(n) = W(n/2) + 1$

Example: Recursive Binary Search, Stage 2

Backwards substitution:

$W(1) = 1$

$W(n) = W(n/2) + 1$

$W(n) = W((n/2)/2) + 1 + 1$ (substitute)

$W(n) = W(n/4) + 1 + 1$ (simplify)

$W(n) = W((n/4)/2) + 1 + 1 + 1$ (substitute)

$W(n) = W(n/8) + 1 + 1 + 1$ (simplify)

...

$W(n) = W(\frac{n}{2^i}) + i$ (generalize)

Solve for i that results in initial condition:

$\frac{n}{2^i} = 1$

$n = 2^i$

$i = \log_2 n$

Substitute $\log_2 n$ for i: $W(n) = \log_2 n + 1$

Example: Recursive Binary Search, Stage 3

Applying recurrence:
$W(1) = 1$
$W(2) = W(1) + 1 = 1 + 1 = 2$
$W(4) = W(2) + 1 = 2 + 1 = 3$

Applying solution:
$W(1) = \log_2 1 + 1 = 0 + 1 = 1$
$W(2) = \log_2 2 + 1 = 1 + 1 = 2$
$W(4) = \log_4 4 + 1 = 2 + 1 = 3$

Another Example

(Assume `items` starts with an even length.)

```
def fun(items):
    if len(items) <= 1:
        return basic_operation()
    else:
        basic_operation()
        return fun(items[2:]) # slice size is n-2
```

STAGE 1

```python
def fun(items):
    if len(items) <= 1:
        return basic_operation()
    else:
        basic_operation()
        return fun(items[2:]) # slice size is n-2
```

Initial Conditions:

$$T(0) = 1$$

$$T(1) = 1$$

Recursive part:

$$T(n) = T(n-2) + 1$$

STAGE 2

Substitution:

$T(n) = T(n-2) + 1$

$$T(n) = T((n-2)-2) + 1 + 1 \qquad \text{(substitute)}$$
$$T(n) = T(n-4) + 1 + 1 \qquad \text{(simplify)}$$

$$T(n) = T((n-4)-2) + 1 + 1 + 1 \qquad \text{(substitute)}$$
$$T(n) = T(n-6) + 1 + 1 + 1 \qquad \text{(simplify)}$$

...

$T(n) = T(n-2i) + i$

STAGE 2 (Continued)

Recursive term disappears when $n - 2i = 0$ or $n - 2i = 1$ (The first will apply for even n, the second will apply for odd n.)
$n - 2i = 0$
$i = n/2$

Substitute $n/2$ for i:
$T(n) = n/2 + 1$
For even n.

Similarly, $T(n) = \frac{n-1}{2} + 1$ for odd n.

Recurrence Exercise

```
def fun(items):
    n = len(items)
    if n <= 1:
        return 3
    for i in range(4):
        mid = n // 2
        fun(items[:mid])
    sum = 0
    for i in range(n):
        for j in range(n):
            sum = items[i] + items[j]
    return sum
```

Recurrence

$$T(1) = 0$$

$$T(n) = 4T(n/2) + n^2$$

Solving With Backward Substitution

$T(1) = 0$
$T(n) = 4T(\frac{n}{2}) + n^2$

$T(n) = 4(4T(\frac{n}{2}/2) + (\frac{n}{2})^2) + n^2$ (substitute)
$T(n) = 16T(\frac{n}{4}) + n^2 + n^2$ (simplify)

$T(n) = 16(4T(\frac{n}{4}/2) + (\frac{n}{4})^2) + n^2 + n^2$ (substitute)
$T(n) = 64T(\frac{n}{8}) + n^2 + n^2 + n^2$ (simplify)

...
$T(n) = 4^i T(\frac{n}{2^i}) + i \times n^2$ (generalize)
Solve for i that results in initial condition:
$\frac{n}{2^i} = 1$
$i = \log_2 n$
Substitute $\log_2 n$ for i: $T(n) = n^2 \log_2 n$

Checking The Answer

Applying recurrence:

$T(1) = 0$

$T(2) = 2^2 + 4T(1) = 4 + 4(0) = 4$

$T(4) = 4^2 + 4T(2) = 16 + 4(4) = 32$

Applying solution:

$T(1) = 1^2 \times \log_2 1 = 1 \times 0 = 0$

$T(2) = 2^2 \times \log_2 2 = 4 \times 1 = 4$

$T(4) = 4^2 \times \log_2 4 = 16 \times 2 = 32$