Lecture #20: Complexity, Continued

Reprise from Last Time

```
def near(L, x, delta):
    """True iff X differs from some member of sequence L by no
    more than DELTA."""
    for y in L:
        if abs(x-y) <= delta:
            return True
    return False
```

- Would really like $C_{\mathrm{near}}(L, x, delta)$, the cost of computing `near(L, x, delta)`.
- But this is very complicated, with so many messy details that result is not all that useful.
- So settle for $C_{\mathrm{wc}}(N)$, the cost of computing `near(L, x, delta)` for `L` of size N *in the worst case*.

Best of the Worst

- From last time, $C_{\mathrm{wc}}(N) \in O(N)$.
- But in addition, it's also clear that $C_{\mathrm{wc}}(N) \in \Omega(N)$.
- So we can say, most precisely, $C_{\mathrm{wc}}(N) \in \Theta(N)$.
- Generally, when a worst-case time is not $\Theta(\cdot)$, it indicates either that
  - We don't know (haven't proved) what the worst case really is, just put limits on it,
    * Most often happens when we talk about the worst-case for a *problem:* "what's the worst case for the best possible algorithm?"
  - Or we know what the worst-case time is, but it's messy, so we settle for approximations that are easier to deal with.

Example: A Nested Loop

- Consider:

```
def are_duplicates(L):
    for i in range(len(L)-1):
        for j in range(i+1, len(L)):
            if L[i] == L[j]:
                return True
    return False
```

- What can we say about $C(L)$, the cost of computing `are_duplicates` on `L`?
- How about $C_{\text{wc}}(N)$, the worst-case cost of running `are_duplicates` over all sequences of length N?

Example: A Nested Loop (II)

- Ans: Worst case is no duplicates. Outer loop runs `len(L)-1` times. Each time, the inner loop runs `len(L)-i-1` times. So total time is proportional to $(N-2)+(N-3)+\ldots+1 = (N-1)(N-2)/2 \in \Theta(N^2)$, where $N = N(L)$ is the length of L.
- Best case is first two elements are duplicates. Running time is $\Theta(1)$ (i.e., bounded by constant).
- So, $C(L) \in O(N(L)^2)$, $C(L) \in \Omega(1)$,
- And $C_{\mathrm{wc}}(N) \in \Theta(N^2)$.

Example from Homework Question

- Why is this slow (in the `Link` class)?

```
k = 1
p = self
# Find last element of
while k < len(self):
    p = p.rest
```

- How slow is it?

Example: A Tricky Nested Loop

- What can we say about this one (assume `pred` counts as one constant-time operation.)

```
def is_unduplicated(L, pred):
    """True iff the first x in L such that pred(x) is not
    a duplicate. Also true if there is no x with pred(x)."""
    i = 0
    while i < len(L):
        x = L[i]
        i += 1
        if pred(x):
            while i < len(L):
                if x == L[i]:
                    return False
                i += 1
    return True
```

Example: A Tricky Nested Loop (II)

- In this case, despite the nested loop, we read each element of `L` at most once.
- Best case is that `pred(L[0])` and `L[0]=L[1]`.
- So $C(L) \in O(N(L))$, $C(L) \in \Omega(1)$.
- And $C_{\mathrm{wc}}(N) \in \Theta(N)$.

Fast Growth

- Consider `Hackenmax` (a function from a test some semesters ago):

```
def Hakenmax(board, X, Y, N):
    if N <= 0:
        return 0
    else:
        return board(X, Y) \
               + max(Hakenmax(board, X+1, Y, N-1),
                     Hakenmax(board, X, Y+1, N-1))
```

- Time clearly depends on `N`. Counting calls to `board`, $C(N)$,the cost of calling `Hackenmax(board,X,Y,N)`, is

$$C(N) = \begin{cases} 0, & \text{for } N \leq 0 \\ 1 + 2C(N-1), & \text{otherwise.} \end{cases}$$

- Using simple-minded expansion,

$$C(N) = 1+2C(N-1) = 1+2+4C(N-2) = \ldots = 1+2+4+8+\ldots+2^{N-1} \in \Theta(2^N).$$

Slow Growth

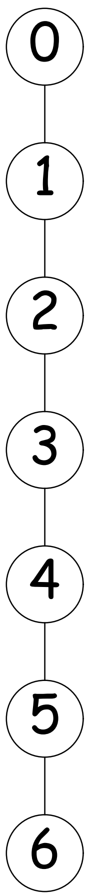
Consider a problem with this structure:

```
def tree_find(T, disc):
    p = disc(T.label)
    if p == -1:
       return T.label
    elif T.is_leaf():
       return None
    else:
       return tree_find(T.children[p], disc)
```

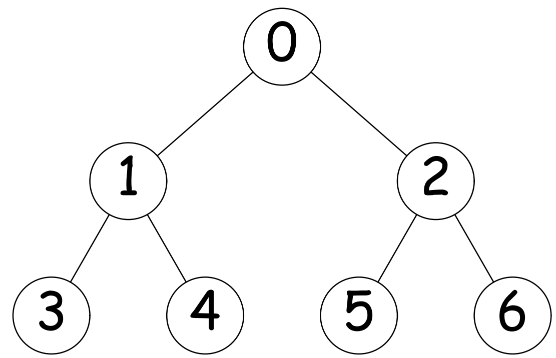
Assume that function `disc` takes (no more than) a constant amount of time.

Kinds of Tree

- Assume we are dealing with binary trees (number of children ≤ 2).
- Trees could have various shapes, which we can classify as "shallow" (or "bushy") and "stringy."



Maximally Deep ("Stringy") Tree



Maximally Shallow ("Bushy") Tree

Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children ≤ 2)?
  - 1. As a function of D, the depth of the tree?
  - 2. As a function of N, the number of keys in the tree?
  - 3. As a function of D if the tree is as shallow as possible for the amount of data?
  - 3. As a function of N if the tree is as shallow as possible for the amount of data?

Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children ≤ 2)?
  - 1. As a function of D, the depth of the tree? $\Theta(D)$
  - 2. As a function of N, the number of keys in the tree?
  - 3. As a function of D if the tree is as shallow as possible for the amount of data?
  - 3. As a function of N if the tree is as shallow as possible for the amount of data?

Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children ≤ 2)?
  - 1. As a function of D, the depth of the tree? $\Theta(D)$
  - 2. As a function of N, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of D if the tree is as shallow as possible for the amount of data?
  - 3. As a function of N if the tree is as shallow as possible for the amount of data?

Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children ≤ 2)?
  - 1. As a function of D, the depth of the tree? $\Theta(D)$
  - 2. As a function of N, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of D if the tree is as shallow as possible for the amount of data? $\Theta(D)$
  - 3. As a function of N if the tree is as shallow as possible for the amount of data?

Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children ≤ 2)?
  - 1. As a function of D, the depth of the tree? $\Theta(D)$
  - 2. As a function of N, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of D if the tree is as shallow as possible for the amount of data? $\Theta(D)$
  - 3. As a function of N if the tree is as shallow as possible for the amount of data? $\Theta(\lg N)$

Questions

- How long does the `tree_find` program (search a tree) take in the worst case on a binary tree (number of children ≤ 2)?
  - 1. As a function of D, the depth of the tree? $\Theta(D)$
  - 2. As a function of N, the number of keys in the tree? $\Theta(N)$
  - 3. As a function of D if the tree is as shallow as possible for the amount of data? $\Theta(D)$
  - 3. As a function of N if the tree is as shallow as possible for the amount of data? $\Theta(\lg N)$

Some Useful Properties

- We've already seen that $\Theta(K_0 N + K_1) = \Theta(N)$ (K, k, K_i here and elsewhere are constants).
- $\Theta(N^k + N^{k-1}) = \Theta(N^k)$. Why?
- $\Theta(|f(N)| + |g(N)|) = \Theta(\max(|f(N)|, |g(N)|))$. Why?
- $\Theta(\log_a N) = \Theta(\log_b N)$. Why? (As a result, we usually use $\log_2 N = \lg N$ for all logarithms.)
- Tricky: why *isn't* $\Theta(f(N) + g(N)) = \Theta(\max(f(N), g(N)))$?
- $\Theta(N^{k_1}) \subset \Theta(k_2^N)$, if $k_2 > 1$. Why?