CSC236 Week 10: Recurrences...

Hisbaan Noorani

November 18 – November 24, 2021

Contents

1 Recursive definition

This sequence comes up in applied rabbit breeding, the height of AVL tress, and the complexity of Euclid's algorithm for the GCD, and an astonishing number of other places: Define for n a natural number,

$$F(n) = \begin{cases} n & n < 2 \\ F(n-2) + F(n-1) & n \geq 2 \end{cases}$$

1.1 Fibonacci patterns

What are Fibonacci numbers multiples of? Which Fibonacci numbers are multiples of 5?

Proof: We define the predicate $P(n)$ for all natural numbers n to be: $\exists k \in \mathbb{N}$ s.t. $F(5n) = 5k$. We will prove, using simple induction that $\forall n \in \mathbb{N}, P(n)$ (Note: for every $n, F(5n+2), F(5n+3)$ are recursively defined).

- Base Case ($n = 0$): $F(5 \cdot 0) = F(0) = 05 \cdot 0$, so $P(0)$ holds,

- Inducvite step: Let $n \in \mathbb{N}$. Assume $P(n)$ i.e. $\exists k \in \mathbb{N}$ s.t. $F(5n) = 5k$. It suffices to prove that $\exists k' \in \mathbb{N}$ s.t. $F(5(n+1)) = 5k'$

  Let $k' = (3k + F(5n+1))$. Then,

$$\begin{aligned} F(5(n+1)) &= F(5n+5) \\ &= F(5n+3) + F(5n+4) \\ &= F(5n+2) + F(5n+3) + F(5n+3) \\ &= F(5n+1) + F(5n+1) + F(5n+2) + F(5n+2) + F(5n+2) \\ &= 3F(5n) + 5F(5n+1) \\ &= 3 \cdot 5k + 5F(5n+1) && \text{(By } P(n)\text{)} \\ &= 5 \cdot (3k + F(5n+1)) \\ &= 5k' \end{aligned}$$

  So $P(n+1)$ follows from $P(n)$

So $\forall \in \mathbb{N}, P(n)$

1.2 Closed form for $F(n)$?

No rabbit, no hat

The course notes present a proof by induction that

$$F(n) = \frac{\phi^n - \left(\hat{\phi}\right)^n}{\sqrt{5}}, \phi = \frac{1+\sqrt{5}}{2}, \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

You can, and should, be able to ...

2 Binary search

The binary search function below is represented by the function $T(n)$.

Since we are not proving correctness, we don't care whether A is sorted. $|A| = n = e - b + 1$, where n is the length of the string/list/tree, b is the beginning of the search, and e is the end of the search.

```
1  def recBinSearch(x, A, b, e):
2      if b == e:              #
3          if x <= A[b]:       #
4              return b        # has time complexity c
5          else:               #
6              return e + 1    #
7      else:
8          m = (b + e) // 2 # midpoint  # has time complexity 1
9          if x <= A[m]:
10             return recBinSearch(x, A, b, m)      # has time complexity
11         else:                                    # max{T(⌊n/2⌋), T(⌈n/2⌉)}
12             return recBinSearch(x, A, m + 1, e)  #
```

Left as an exercise to the reader: Show $\left\lceil \frac{n}{2} \right\rceil = m - b + 1$. $\left\lfloor \frac{n}{2} \right\rfloor = e - m$

2.1 Guess the bound of $T(n)$ by "unwinding it"

Suppose (thoughtful wishing...) that $n = 2^k$ for some natural number k.

$$\begin{aligned}
T(n) &= T(2^k) \\
&= 1 + T(2^{k-1}) \\
&= 1 + 1 + T(2^{k-2}) \\
&= 1 + 1 + 1 + T(2^{k-3}) \\
&\quad \cdots \\
&= k + T(1) \\
&= \log_2 n + c && T(1) = C \text{ and } 2^k = n \text{ so, } \log_2 n = k)
\end{aligned}$$

This is a guess! Even for powers of 2, there's an induction (on k maybe?) here.

2.2 Prove lower bound on $T(n)$

Proof: Let $B \in \mathbb{R}^+ = 2$. Let $c \in \mathbb{R}^+ = 1$. Let $n \in \mathbb{N}$. Assume $n \geq B$. Assume $\forall i$ s.t. $B \leq i < n, T(i) \geq c \cdot \log_2(i)$.

- Case $n \geq 3$:

$$\begin{aligned}
T(n) &= 1 + \max\left\{T\left(\left\lceil \frac{n}{2} \right\rceil\right), T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right\} \\
&\geq 1 + c\log_2\left(\left\lceil \frac{n}{2} \right\rceil\right) && \text{(By IH, since } n > \left\lceil \frac{n}{2} \right\rceil \geq B \text{ since } B \geq 2) \\
&\geq 1 + c\log_2\left(\frac{n}{2}\right) && \text{(since } \frac{n}{2} \leq \left\lceil \frac{n}{2} \right\rceil) \\
&= 1 + c\left(\log_2(n) - \log_2(2)\right) \\
&= 1 + c\left(\log_2(n) - 1\right) \\
&= (1 - c) + c\log_2(n) \\
&\geq c\log_2(n) && \text{(since } c \leq 1)
\end{aligned}$$

- Case $n = 2$: $T(2) = 1 + T(1) = 1 + c' \geq 1 \cdot \log_2(2) = c\log_2(2)$
- Case $n = 1$: $T(1) = c' \geq 1 \cdot \log_2(1) = 0 = c \cdot 0 = c\log_2(1)$

So the lower bound of $T(n)$ is $c\log_2(n)$ ■

2.3 Prove upper bound on $T(n)$

Proof: Let $B \in \mathbb{R}^+ = \ldots$. Let $c \in \mathbb{R}^+ = \ldots$ Let $n \in \mathbb{N}$. Assume $n \geq B$ and that $T(i) \leq c \cdot \log_2 i$ for all $B \leq i < n$.

It suffices to show that $T(n) \leq c\log_2(n)$

- Case $n = \ldots$:

$$\begin{aligned}
T(n) &= 1 + \max\left\{T\left(\left\lceil \frac{n}{2} \right\rceil\right), T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right\} \\
&\leq 1 + c\log_2\left(\left\lceil \frac{n}{2} \right\rceil\right) && \text{(since } B \leq \left\lceil \frac{n}{2} \right\rceil < n \text{, since } n \geq 2\text{)} \\
&\leq 1 + c\log_2\left(\frac{n+1}{1}\right) \\
&= 1 + c(\log_2(n+1) - \log_2(2)) \\
&= 1 + c(\log_2(n+1) - 1) \\
&= (1-c) + c\log_2(n+1)
\end{aligned}$$

  Oops! We want $T(n) \leq c\log_2(n)$, not just $T(n) \leq c\log_2(n+1)$. We could fiddle aroudn and make this work, by strenghtening the claim to:

$$T(n) \leq c\log_2(n-1) \leq c\log_2(n)$$

  This strengthens the IH, and it should work. Howerver, will it generalize to other recursive algorithms that use the "divide and conquer" approach? ■

There are other strategies that will work to prove this.

3 Reucrrence for MergeSort

```
1  MergeSort(A, b, e) -> None:
2      if b == e: return None  # c_1
3      m = (b + e) / 2
4
5      MergeSort(A, b, m)      #
6      MergeSort(A, m + 1, e)  #
7
8      # merge sorted A[b..m] and A[m + 1..e] back into A[b..e]
9      B = A[:] # copy A  # linear in n              #
10     c = b                                          #
11     d = m + 1                                      #
12     for i in [b,...,e]:                            #
13         if d > e or (c <= m and B[c] < B[d]):      #
14             A[i] = B[c]                            # linear in n
15             c = c + 1                              #
16         else: # d <= e and (c > m or B[c] >= B[d]) #
17             A[i] = B[d]                            #
18             d = d + 1                              #
```

3.1 Unwind (repeated substitution)

$T(n) = 2T\left(\frac{n}{2}\right) + n$

Assume $n = 2^k$ for some positive integer k.

$$
\begin{aligned}
T(n) &= T(2^k) \\
&= T\left(\left\lceil 2^{k-1} \right\rceil\right) + T\left(\left\lfloor 2^{k-1} \right\rfloor\right) + n && \text{(by definition)} \\
&= T\left(2^{k-1}\right) + T\left(2^{k-1}\right) + 2^k && \text{(since } 2^k \text{ is always whole)} \\
&= 2T\left(2^{k-1}\right) + 2^k \\
&\quad \ldots && \text{(repeat } k-1 \text{ more times)} \\
&= 2^k T(1) + k\left(2^k\right) \\
&= n \cdot c + \log_2(n) \cdot n && \text{(since } n = 2^k \text{ and } T(1) = c) \\
&= cn + n\log_2(n)
\end{aligned}
$$

So for the case $n = 2^k, T(n) = cn + n\log_2(n)$, which can be simplified to $T(2^k) = 2^k c + 2^k k$ (only for this specific case).