Recurrent Problems

Madiba Hudson-Quansah

September 2023

Contents

Chapter 1

Introduction

Problems that have been investigated repeatedly by mathematicians and whose solutions use the idea of recurrence, i.e. the solution to each problem depends on the solutions to smaller instances of the same problem.

1.1 Recurrence Relations

1.1.1 Writing Recurrences from algorithms

Algorithm 1 recurse1 (n)

```
if $n < 3$ then
    return 80
end if
for $i \leftarrow 0$ to $n$ do
    PRINT(i)
end for
v1 ← RECURSE1( $n/3$)
v2 ← RECURSE1( $n/3$)
return v1 + v2
```

We first:

- Find the time taken by the base case
- Find the time taken by the recursive calls

∴

$$T(n) = \begin{cases} 2 & \text{if } n < 3 \\ n + 2T(n/3) + 2 & \text{otherwise} \end{cases}$$

Algorithm 2 factorial (n)

```
if $n = 1$ then
    return 1
end if
return $n\times$ FACTORIAL( $n-1$)
```

$$T(n) = \begin{cases} 2 & \text{if } n = 1 \\ T(n-1) + 1 & \text{if } n > 1 \end{cases}$$

Question 1

Prove by indunction that $n! > 3^n$ for $n \geqslant 7$

Solution:

Basis Step

$$\begin{aligned} P(n) &: n! > 3^n \\ P(7) &: 7! > 3^7 \\ &: 5040 > 2187 \end{aligned}$$

Induction Step

Assume $P(k)$ is true, i.e.:

$$P(k) : k! > 3^k$$

Now we have to prove $P(k+1)$ is true:

$$\begin{aligned} P(k+1) &: (k+1)! > 3^{k+1} \\ &: k!(k+1) \\ &> 3^k(k+1) \\ &> 3^k(7+1) \\ 3 \times 3^k &> 3^{k+1} \end{aligned}$$

$\therefore P(k+1)$ is T

Question 2

Prove by induction that $6^n - 1$ is divisible by 5, for $n \geqslant 1$

Solution:

Basis Step

$$\begin{aligned} P(1) &: 6^n - 1 = 5k,\ k \in \mathbb{Z} \\ &: 6 - 1 = 5k \\ &: 5(1) = 5k \\ k &= 1 \end{aligned}$$

Induction Step

Assume $P(k)$ is true, i.e.:

$$P(k) : 6^k - 1 = 5t,\ t \in \mathbb{Z}$$

Then $P(k+1)$:

$$\begin{aligned}
P(k+1) &: 6^{k+1} - 1 \\
&: 6\left(6^k\right) - 1 \\
&: 6\left(6^k\right) - 6 + 5 \\
&: 6\left(6^k - 1\right) + 5 && \text{Let } u = 6^k - 1 \\
&: 6u + 5
\end{aligned}$$

$6u$ is divisible by 5 because u is divisible by 5 and 5 is divisible by 5 so $P(k+1)$ is T

1.2 Tower of Hanoi

Invented by French mathematician Edouard Lucas in 1883. The problem consists of a tower of eight disks initially stacked in decreasing size on one of three pegs. The objective is to transfer the entire tower to one of the other pegs, moving only one disk at a time and never moving a larger one onto a smaller.

1.2.1 Look at small cases

LOOK AT SMALL CASES - The first step in solving this is to generalize the question, i.e Let n be the number of disks initially stacked.

NAME AND CONQUER - The next step is to introduce appropriate notation, i.e. Let T_n be the minimum number of moves that will transfer n disks from one peg to another. Therefore $T_0 = 0$, $T_1 = 1$ and $T_2 = 3$.

The next step is to use this generalization to scale to different disk numbers. Through trial and error we can see that the best strategy for transferring n disks is to first transfer the $n-1$ smallest disk to a different peg, requiring T_{n-1} moves. Then move the largest disk to the last peg, also requiring one move. Then finally transfer the $n-1$ smallest back onto the third peg where the largest disk already is, requiring another T_{n-1} moves.

1.2.2 Find and prove a mathematical expression

Thus we can transfer n disks (where $n > 0$) in at most $2T_{n-1} + 1$ moves, i.e. :

$$T_n \leqslant 2T_{n-1} + 1 \quad n > 0$$

$\leqslant$ instead of = due to our construction only proving that $2T_{n-1} + 1$ moves work, not that all $2T_{n-1} + 1$ moves are necessary. The however is the optimal solution, due to us having to move the largest disk at least once and having to have the $n-1$ smallest disk be alone on one peg every time we do move the largest disk, requiring T_{n-1} moves every time.

The result factoring in the base condition of $T_0 = 0$ is:

$$\begin{gathered}
T_0 = 0 \\
T_n = 2T_{n-1} + 1 \quad n > 0
\end{gathered}$$

These sets of in equalities is called a recurrence / recurrence relation / recursion relation. It gives a boundary value and an equation for the general value in terms of the earlier ones.

Computing from recurrence for a large value of n takes too long so we need a *closed form* for T_n that lets us compute large ns quickly. With a closed form we can understand what T_n really is.

1.2.3 Find and prove a closed form for our expression

One way of solving a recurrence is to guess a solution, them prove our guess is correct. To do this we look again at small cases. By successively computing T_n:

$$\begin{aligned} T_3 &= 2 \times 3 + 1 = 7 \\ T_4 &= 2 \times 7 + 1 = 15 \\ T_5 &= 2 \times 15 + 1 = 31 \\ T_6 &= 2 \times 31 + 1 = 63 \end{aligned}$$

We can see it looks like: when $n \leqslant 6$

$$T_n = 2^n - 1 \quad n \geqslant 0$$

This technique is called *Mathematical induction.* First we prove the *basis* (when n is at its least i.e. n_0), then we prove the statement for $n > n_0$ assuming that it has already been proved for all values between n_0 and $n - 1$ inclusive, i.e. the *induction.*

Applying this to our recurrence:

$$T_n = 2T_{n-1} + 1$$

$$T_n = 2^n - 1$$

$$\begin{aligned} T_n &= 2(2^{n-1} - 1) + 1 \\ T_n &= 2^n - 2 + 1 \\ &\therefore \\ T_n &= 2^n - 1 \end{aligned}$$

The Tower of Hanoi highlights the steps needed to find a closed-form expression for some quantity of interest:

- Look at small cases (1.2.1).
- Find and prove a mathematical expression for he quantity of interest (1.2.2).
- Find and prove a closed form for our mathematical expression (1.2.3).