Recursion (II): Binary Recursion

CSCD 300 – Data Structures

Eastern Washington University

Outline

1. Binary recursion

2. Example: Check the uniqueness of an array of numbers

Binary recursion

A binary recursion is one that has two recursive function calls inside the function body.

Example: Check the uniqueness of an array of numbers

Problem statement

- Input: an array of numbers.
- Output: `true`, if the array of numbers are all unique; `false`, otherwise.

Examples

- Input: $\{4, 3, 5, 2, 3, 7, 7\} \Rightarrow$ Output: *false*
- Input: $\{4, 3, 5, 2, 7, 8, 9\} \Rightarrow$ Output: *true*

A recursive idea

$$A[0 \ldots n-1] \text{ is unique.} \iff \begin{cases} A[0 \ldots n-2] \text{ is unique.} \\ \qquad \text{AND} \\ A[1 \ldots n-1] \text{ is unique.} \\ \qquad \text{AND} \\ A[0] \neq A[n-1] \end{cases}$$

The code follows ...

```
/* Decide whether all the array elements in A[low ...high] are unique.
   Assume: A.length = n
*/
boolean isUnique(int[] A, int low, int high)
{
  if(low == high) return true;

  if(isUnique(A, low, high-1) == false)
     return false;
  if(isUnique(A, low+1, high) == false)
     return false;

  if(A[low] == A[high])
     return false;

  return true;
}
```

The function call `isUnique(A, 0, n-1)` checks whether all the numbers A is unique.

Time cost analysis

- Let $x = high - low + 1$ denote the input size.
- Let $T(x)$ represents the time cost of `isUnique(A, low, high)`

```
boolean isUnique(int[] A, int low, int high)  -----> T(x)
{
  if(low == high) return true;     -------> constant time cost: c1

  if(isUnique(A, low, high-1) == false)       ------> T(x-1)
     return false;
  if(isUnique(A, low+1, high) == false)       ------> T(x-1)
     return false;

  if(A[low] != A[high])            ----------+
     return true;                            |  costant time cost: c2
                                             |
  return false;                    ----------+
}
```

$$T(x) = c_1 + c_2 + 2T(x-1) = c + 2T(x-1) \quad (\textit{continue...})$$

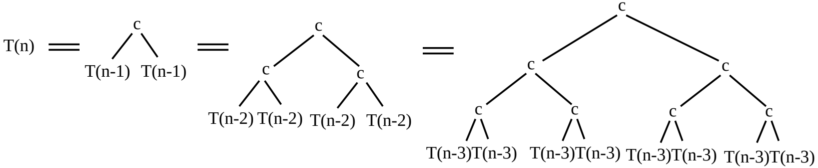
So the time cost of `isUnique(A, 0, n-1)` is ...

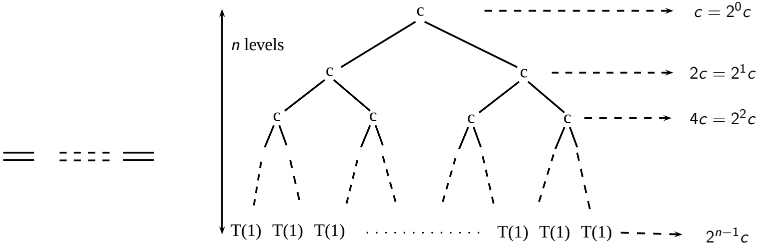
$$T(n) = c + 2T(n-1)$$

Recursively,

$$\begin{aligned}
T(n-1) &= c + 2T(n-2) \\
T(n-2) &= c + 2T(n-3) \\
T(n-3) &= c + 2T(n-4) \\
&\vdots \\
T(2) &= c + 2T(1) \\
T(1) &= c
\end{aligned}$$

Let's visualize the decomposition of $T(n)$ ...





Note that $T(1)$ is also a constant

$$\begin{aligned} T(n) &= \text{the summation of all the nodes in the final tree} \\ &= 2^0c + 2^1c + 2^2c + \ldots + 2^{n-1}c \end{aligned}$$

(continue ...)

$$
\begin{aligned}
T(n) &= \text{the summation of all the nodes in the final tree} \\
&= 2^0c + 2^1c + 2^2c + \ldots + 2^{n-1}c = \underbrace{\left(2^0 + 2^1 + 2^2 + \ldots + 2^{n-1}\right)c}_{\text{the summation of a geometric series}} \\
&= (2^n - 1)c = c2^n - c = O\left(2^n\right)
\end{aligned}
$$

indicating this program will be extremely slow even when the input size n becomes not so large. (You can implement the program and test it with a reasonably sized array, say 100000. The program will never finish.)

Background knowledge: geometric series

The summer of a geometric series can be calculated by using the following formula [a]: If $r \neq 1$,

$$r^0 + r^1 + r^2 + \ldots + r^{n-1} = \frac{1 - r^n}{1 - r}$$

[a] http://en.wikipedia.org/wiki/Geometric_series

Questions for you to think

Question 1: What is the exact reason that causes this recursion-based program so slow ?

Open discussion with your classmates.

Question 2: What is the efficient solution ?

You will be able to design an efficient solution using the dynamic programming strategy which will be discussed in CSCD320.