COMP 250

Lecture 33

recurrences 1

Nov. 26, 2018

What's left to do ?

- Lecture 33, 34 : Recurrences
- Lecture 35, 36, 37: Asymptotic Complexity

Let $t(n)$ be the time or number of instructions to execute an algorithm.

It is relatively easy to determine $t(n)$ when our algorithms only have loops:

e.g. addition and multiplication,

manipulating a list,

quadratic sorting, …

But how do we determine a $t(n)$ for a recursive algorithm ?

Example: Suppose a list has n elements. What is $t(n)$ for the following?

```
reverse( list ){
   if list.size == 1
      return  list
   else{
      firstElement  =  list.removeFirst( )
      list   =  reverse( list )
      return   list.addLast( firstElement )
      }
 }
```

Recurrence Relation

A recurrence relation is a sequence of numbers where the n-th term depends on previous terms.

e.g. Fibonacci $F(n) = F(n-1) + F(n-2)$

We will consider recurrence relations for $t(n)$, the time to execute a recursive algorithm, as a function of the *input size* n. The recurrence expresses it in terms of the smaller input size.

Recurrence Relation

A recurrence relation is a sequence of numbers where the n-th term depends on previous terms.

e.g. Fibonacci $F(n) = F(n-1) + F(n-2)$

We will consider recurrence relations for $t(n)$, the time to execute a recursive algorithm, as a function of the *input size* n. The recurrence expresses it in terms of the smaller input size.

Note: for Fibonacci numbers, n is an input value. It is NOT the input size. See Exercises.

Example 1: Reversing a list

```
reverse( list ){
      if list.size == 1
         return  list
      else{
         firstElement  =  list.removeFirst( )
         list   =  reverse( list )
         return   list.addLast( firstElement )
       }
}
```

Base case $n = 1$

$$t(n) = c + t(n-1)$$

Q: What assumptions are we making about removeFirst() and addLast() here ?

Q: What assumptions are we making about removeFirst() and addLast() here ?

A: They can be done in constant time.
(The former is not true if we use an array list.)

Solving a recurrence using back substitution

$$t(n) \quad = \quad c + t(n-1)$$

Solving a recurrence using back substitution

$$\begin{aligned} t(n) &= c + t(n-1) \\ &= c + c + t(n-2) \end{aligned}$$

Solving a recurrence using back substitution

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

$$
\begin{aligned}
t(n) &= c + t(n-1) \\
&= c + c + t(n-2) \\
&= c + c + c + t(n-3) \\
&= \ldots \\
&= c\ (n-1) + t(1)
\end{aligned}
$$

if base case is $n = 1$
(reversing a list)

Solving a recurrence using back substitution

$$
\begin{aligned}
t(n) &= c + t(n-1) \\
&= c + c + t(n-2) \\
&= c + c + c + t(n-3) \\
&= \ldots \\
&= cn + t(0)
\end{aligned}
$$

if base case is $n = 0$

Sorting a list

```
sort( list )  {
     if list.size == 1
        return  list
     else{
         minElement = list.removeMin()
         list = sort(list)
         return list.addFirst( minElement )
      }
}
```

What is the recurrence relation?

Sorting a list

```
sort( list ) {
    if list.size == 1
       return  list
    else{
        minElement = list.removeMin()
        list = sort(list)
        return list.addFirst( minElement )
     }
}
```

Base case $n = 1$

$$t(n) = c_1 + \boxed{c_2\, n} + \boxed{t(n-1)}$$

Q: What assumptions are we making about addFirst() here ?

A: It is ok, if this step uses time proportional to n. Why?

Q: What assumptions are we making about addFirst() here ?

A: It is ok, if this step uses time proportional to n. Why?

Because we listRemove() already has time proportional to n.

Let’s solve the slightly simpler recurrence.

$$t(n) \quad = \quad c\,n + t(n-1)$$

$$
\begin{aligned}
t(n) &= c\,n + t(n-1) \\
&= c\,n + c \cdot (n-1) + t(n-2)
\end{aligned}
$$

$$
\begin{aligned}
t(n) &= c\,n + t(n-1) \\
&= c\,n + c \cdot (n-1) + t(n-2) \\
&= \ldots \\
&= c\,\{\, n + (n-1) + (n-2)\ + \cdots + (n-k)\} + t(n-k-1)
\end{aligned}
$$

Let’s look at the slightly simpler recurrence, **and cleaner base case** $(\boldsymbol{n = 0})$.

$$
\begin{aligned}
t(n) &= c\,n + t(n-1) \\
&= c\,n + c \cdot (n-1) + t(n-2) \\
&= \ldots \\
&= c\,\{\,n + (n-1) + (n-2)\ + \cdots + (n-k)\} + t(n-k-1) \\
&= c\,\{\,n + (n-1) + (n-2)\ + \cdots + 2 + 1\} + t(0)
\end{aligned}
$$

$$n - k = 1$$

Let's look at the slightly simpler recurrence, and cleaner base case $(n = 0)$.

$$
\begin{aligned}
t(n) &= c\,n + t(n-1) \\
&= c\,n + c \cdot (n-1) + t(n-2) \\
&= \ldots \\
&= c\,\{\, n + (n-1) + (n-2)\ + \cdots + (n-k)\} + t(n-k-1) \\
&= c\,\{\, n + (n-1) + (n-2)\ + \cdots + 2 + 1\} + t(0) \\
&= \frac{cn(n+1)}{2} + t(0)
\end{aligned}
$$

Example 3: Tower of Hanoi

```
tower(n, start, finish, other){
    if  n > 0 {
        tower( n-1, start, other,  finish)
        move from start to finish
        tower( n-1, other, finish, start)
    }
}
```

$$t(n) = c + 2\,t(n-1)$$

Example 3: Tower of Hanoi

```
tower(n, start, finish, other){      //  base case is n=0
    if  n > 0 {
          tower( n-1, start, other,  finish)
          move from start to finish
          tower( n-1, other, finish, start)
     }
}
```

$$t(n) = c + 2\,t(n-1)$$

What do you think the solution will be ?

Tower of Hanoi recurrence

$$t(n) \quad = \quad c + 2\ t(n-1)$$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= c + 2\ t(n-1) \\
&= c + 2(c + 2\ t(n-2))
\end{aligned}
$$

Tower of Hanoi recurrence

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

Tower of Hanoi recurrence

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

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= c + 2\ t(n-1) \\
&= c + 2(c + 2\ t(n-2)) \\
&= c\ (1+2) + 4\ t(n-2) \\
&= c\ (1+2) + 4\ (c + 2\ t(n-3)) \\
&= c\ (1+2+4)\ +\ 8\ t(n-3)
\end{aligned}
$$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= c + 2\ t(n-1) \\
&= c + 2(c + 2\ t(n-2)) \\
&= c\ (1+2) + 4\ t(n-2) \\
&= c\ (1+2) + 4\ (c + 2\ t(n-3)) \\
&= c\ (1+2+4)\ +\ 8\ t(n-3) \\
&= \ldots \\
&= c\ (1+2+4+8+\cdots+2^{k-1}) + 2^k\ \ t(n-k)
\end{aligned}
$$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= c + 2\ t(n-1) \\
&= c + 2(c + 2\ t(n-2)) \\
&= c\ (1+2) + 4\ t(n-2) \\
&= c\ (1+2) + 4\ (c + 2\ t(n-3)) \\
&= c\ (1+2+4)\ +\ 8\ t(n-3) \\
&= \dots \\
&= c\ (1+2+4+8+\cdots+2^{k-1}) + 2^k\ \ t(n-k) \\
&= c\ (1+2+4+8+\cdots+2^{n-1}) + 2^n\ t(0)
\end{aligned}
$$

$n = k$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= c + 2\ t(n-1) \\
&= c + 2(c + 2\ t(n-2)) \\
&= c\ (1+2) + 4\ t(n-2) \\
&= c\ (1+2) + 4\ (c + 2\ t(n-3)) \\
&= c\ (1+2+4)\ +\ 8\ t(n-3) \\
&= \ldots \\
&= c\ (1+2+4+8+\cdots+2^{k-1}) + 2^k\ \ t(n-k) \\
&= c\ (1+2+4+8+\cdots+2^{n-1}) + 2^n\ t(0) \\
&= c\ (2^n - 1) + 2^n\ t(0)
\end{aligned}
$$

Base case for Tower of Hanoi

You should know ….

$$1 + 2 + 3 + \ \ldots + \ k \ = \ ?$$

$$1 + 2 + 4 + 8 + \ \ldots + \ 2^k \ = \ ?$$

$$1 + x + x^2 + x^3 + \ \ldots + x^k = \ ?$$

Example 4: Binary Search

```
binarySearch( list, value, low, high ){
    if  low <= high {
        mid = low + (high - low) / 2
        if value == list[mid]
            return value
        else   if value < list[mid]
            return  binarySearch(list, value,  low,     mid - 1 )
        else
            return  binarySearch(list, value,  mid+1,  high)
        }
    else
        return   -1
}
```

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

Example 4: Binary Search

```
binarySearch( list, value, low, high ){
    if  low <= high {
        mid = low + (high - low) / 2
        if value == list[mid]
            return value                          Base case  n = ?
        else   if value < list[mid]
            return  binarySearch(list, value,  low,     mid - 1 )
        else
            return  binarySearch(list, value,  mid+1,  high)
        }
    else
        return   -1                  Base case  n = ?
}
```

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

Example 4: Binary Search

```
binarySearch( list, value, low, high ){
    if  low <= high {
        mid = low + (high - low) / 2
        if value == list[mid]
            return value                          Base case  n = 1
        else   if value < list[mid]
            return  binarySearch(list, value,  low,     mid - 1 )
        else
            return  binarySearch(list, value,  mid+1,  high)
        }
    else
        return   -1                   Base case  n = 0
}
```

$$t(n) = c + t(\frac{n}{2})$$

Suppose $n = 2^k$.

$$t(n) = c + t(n/2)$$

Suppose $n = 2^k$.

$$
\begin{aligned}
t(n) &= c + t(n/2) \\
&= c + c + t(n/4)
\end{aligned}
$$

Suppose $n = 2^k$.

$$
\begin{aligned}
t(n) &= c + t(n/2) \\
&= c + c + t(n/4) \\
&= c + c + \cdots + t(n/2^k)
\end{aligned}
$$

Suppose $n = 2^k$.

$$
\begin{aligned}
t(n) &= c + t(n/2) \\
&= c + c + t(n/4) \\
&= c + c + \cdots + t(n/2^k) \\
&= c + c + \cdots + c + t(n/n)
\end{aligned}
$$

Suppose $n = 2^k$.

$$\begin{aligned} t(n) &= c + t(n/2) \\ &= c + c + t(n/4) \\ &= c + c + \cdots + t(n/2^k) \\ &= c + c + \cdots + c + t(n/n) \\ &= c \log_2 n + t(1) \end{aligned}$$

Base case (we can think of it as including t(0) case)

Today's Recurrences

$$t(n) = c + t(n-1)$$

$$t(n) = c\,n + t(n-1)$$

$$t(n) = c + 2\,t(n-1)$$

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

Announcement

- Quiz 5 is on Friday.   It covers maps & hashing, graphs, and today's lecture.