جامعة طرابلس ـ كلية تقنية المعلومات

Design and Analysis Algorithms

تصميم وتحليل خوارزميات

ITGS301

Lecture 5 : المحاضرة الخامسة

Tripoli University - Faculty of Information Technology
Marwa Solla
m.solla@uot.edu.ly

Algorithm

Iteration

T(n) = sum of computing time of all statements

Closed-end equation

Recursion

T(n) = recurrence equation

Closed-end equation

Classify Order

What is a Recursion ?

Recurrence Relations

Definition: A *recurrence relation* for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more of the previous terms of the sequence, namely, $a_0, a_1, \ldots, a_{n-1}$, for all integers n with $n \geq n_0$, where n_0 is a nonnegative integer.

Recurrence Relations

When an algorithm contains a recursion call to itself, we can often describe the running time by ***recurrence equation* or *recurrence***. The recurrence describes the over all running time on the problem of size ***n*** in terms of the running time on smaller inputs. ***Recurrence*** is an equation that describes a function in term of its value on small inputs

A recurrence is an equation that is used to represent the running time of a recursive algorithm

Recurrence relations result naturally from the analysis of recursive algorithms, solving recurrence relations yields a *closed-end formula for calculation of run time.*

العلاقة التكرارية هي معادلة رياضية تستخدم لتمثل وقت الخوارزميات ذاتية الاستدعاء

Cases of a Recurrence Relations

A recursive algorithm has two cases:
(1) Base Case
(2) Recursive Case

General form of a Recurrence Relations

$$T(n) = \begin{cases} c & n \leq 1 \quad \leftarrow \text{Base Case} \\ aT(n/b) + f(n) & n > 1 \quad \leftarrow \text{Recursive Case} \end{cases}$$

a: the number of times a function calls itself

b: the factor by which the input size is reduced

$f(n)$: the run time of each recursive call

For examples,

Example 1: the recursive Algorithm to compute n! :

```
/* Returns n!= 1*2*3...(n-1)*n for n >= 0. */
int factorial (int n)
 {
   if (n == 1) return 1;
   else
  return factorial (n-1) * n;
}
```

The running time, T(n), can be defined as recurrence equation:

```
T(n) = 1       n=1
T(n) = T(n-1) + 1 for all n>0
```

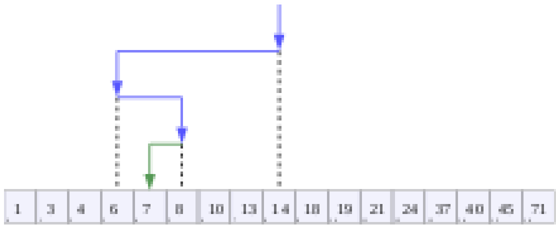
- As we have seen, a *recursive* function is one that is defined in terms of itself.
- Similarly, a *recurrence* is an equation which is defined in terms of itself.
- In the previous example, *factorial* is a recursive function while the time complexity of this function is expressed in terms of a recurrence:

$$n! = \begin{cases} 1 & \text{when } n = 1 \\ n * (n-1)! & \text{otherwise} \end{cases}$$

$$T(n) = \begin{cases} 1 & \text{when } n = 1 \\ T(n-1) + 1 & \text{otherwise} \end{cases}$$

Example 2: (binary search tree) a recursive algorithm to search for X element among n stored elements.

```
ALGORITHM BINARY-SEARCH (A,lo,hi,X)
if (lo > hi)
return FALSE
mid ← ⌊(lo+hi)/2⌋
if x = A[mid]
return TRUE if ( x < A[mid] )
BINARY-SEARCH (A, lo, mid-1, x)
if ( x > A[mid] )
BINARY-SEARCH (A, mid+1, hi, x)
```



The running time, T(n), can be defined as recurrence equation:

$T(n) = 1 \quad n=1$

$T(n) = T(n/2) + 1$ for all $n > 1$

Lo			mid			hi
12	18	20	23	35	44	52
a0	a1	a2	a3	a4	a5	a6

Exercise

```
1) int add (int x)
   {
      if (x == 1) return 5;
    else
      return 1 + add (n-1);
   }
```

Recurrence equation is:

$T(n) = 1 \quad n=1$

$T(n) = T(n-1) + 1$ for all $n>1$

Exercise

```
2) int power ( x , n)
  {
     if (n == 0)
       return 1;
    else
      if (n == 1)
        return x;
     else
       if (n % 2=0)
          return power(x,n/2) * power(x,n/2) ;
       else
          return return x *power(x,n/2) * power(x,n/2) ;
 }
```

Recurrence equation is:

T(n) = 1 n=0
T(n) = 2 n=1
T(n) = 2T(n/2) + 1 for all n>1

Solving Recurrence Relations

There are many methods to solve the recurrence relations, some of them are:

- Iteration method.
- The Master method.
- Recursion tree method.

ITERATION METHOD

Iteration method

Iteration is simply the repetition of processing steps. It is used to computing the running time for any recursive algorithm.

Note: We need to solve the recurrence equation by getting the Closed End formula, then calculation of running time.

We will show how this method works by some examples:

Example 1 (Factorial)

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+1 & \text{for all } n > 0 \end{cases}$$

Answer: Iteration T(n)

1. $T(n) = T(n-1) + 1$

2 Since, $T(n-1) = T(n-1-1) + 1$
$= T(n-2) + 1$

then, $T(n) = T(n-2)+1+1$

$= T(n-2) +2$

3 Since, $T(n-2) = T(n-2-1) +1$

$= T(n-3) +1$

then, $T(n) = T(n-3) +1+2$

$= T(n-3) + 3$

4 Since, $T(n-3) = T(n-3-1) + 1$

$= T(n-4) + 1$

then, $T(n) = T(n-4) +1 + 3$

$= T(n-4) + 4$

n $T(n) = T(n-n) + n$

$= T(0) + n$

$= 1 + n$

The closed end formula: $T(n) = 1 + n$

the running time $T(n) = O(n)$

Example 2 (Binary Search)

Find the closed end formula using the iteration method.

$$T(n) = \begin{cases} 1 & n=1 \\ T(n/2) + 1 & \text{for all } n > 1 \end{cases}$$

answer

1 $T(n) = T(n/2) + 1$

2 Since, $T(n/2) = T(n/4) + 1$

Then, $T(n) = T(n/4) + 1 + 1$

$= T(n/4) + 2$

3 Since, $T(n/4) = T(n/8) + 1$

$T(n/2^2) = T(n/2^3) + 1$

Then, $T(n) = T(n/2^3) + 1+2$

$= T(n/2^3) + 3$

.

.

n $T(n) = T(n/2^k) + k$

Since $T(n) = 1$ suppose that $n/2^k$

$n = 2^k$ $k=\log_2 n$ $k= \lg n$

$T(n) = T(1) + k$

$= T(1) + \lg n$

The closed end formula $= 1 + \lg n$

The running time T(n) is O(lg n).

Example 3:

$$T(n) = \begin{cases} 0 & n=0 \\ 2T(n-1) + 1 & \text{for all } n > 0 \end{cases}$$

answer

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

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

Then $T(n) = 2[2T(n-2) + 1] + 1$

$= 4T(n-2) + 2 + 1$

3 $T(n-2) = 2T(n-3) + 1$

Then $T(n) = 4[2T(n-3) + 1] + 2 + 1$

$= 8T(n-3) + 4 + 2 + 1$

$= 2^3\,T(n-3) + 2^2 + 2 + 1$

.

.

n $T(n) = 2^k\,T(n-k) + 2^{k-1} + 2^{k-2} + \dots + 2^1 + 2^0$

When n=0

$n - k = 0 \rightarrow k = n$

$T(n) = 2^n T(n-n) + 2^{n-1} + 2^{n-2} + ... + 2^1 + 2^0$

$= 2^n . T(0) + \sum_{0}^{n-1} 2^k$
$= 2^n.0 + [2^{n-1+1} -1/2-1]$
$= 2^n.0 \quad + [2^n -1]$
$= 2^n -1$

Geometric series: $\sum_{k=0}^{n} x^k = 1 + x + x^2 + ... + x^n = \frac{x^{n+1}-1}{x-1} (x \neq 1)$

The closed end formula = $2^n -1$

The running time = $O(2^n)$

The End .