Mizpah Christian School - https://www.mizpahchristianschool.org

---

Topic :Summations and Recurrences

Summations and Recurrences

When analyzing running time costs for programs with loops, we need to add up the costs for each time the loop is executed.

This is an example of a summation.

Summations are typically written with the following “Sigma” notation:

$$\sum_{i=1}^{N} f(i)$$

or, it can be written as :

$$f(1) + f(2) + \cdots + f(n-1) + f(n).$$

The following is a list of useful summations, along with their closed-form solutions.

$$\sum_{i=1}^{N} n(n+1)/2$$

The running time for a recursive algorithm is most easily expressed by a recursive expression because the total time for the recursive algorithm includes the time to run the recursive call(s). A **recurrence relation** defines a function by means of an expression that includes one or more (smaller) instances of itself.

A classic example is the recursive definition for the factorial function:
$n! = (n - 1)! \cdot n$ for $n > 1$; $1! = 0! = 1$.

Another standard example of a recurrence is the Fibonacci sequence:
$Fib(n) = Fib(n - 1) + Fib(n - 2)$ for $n > 2$; $Fib(1) = Fib(2) = 1$.

Example If we expand the recurrence $T(n) = T(n - 1) + 1$, we get
$T(n) = T(n - 1) + 1$
$= (T(n - 2) + 1) + 1.$
We can expand the recurrence as many steps as we like, but the goal is to detect some pattern that will permit us to rewrite the recurrence in terms of a summation. In this example, we might notice that
$(T(n - 2) + 1) + 1 = T(n - 2) + 2$
and if we expand the recurrence again, we get
$T(n) = T(n - 2) + 2 = T(n - 3) + 1 + 2 = T(n - 3) + 3$
which generalizes to the pattern $T(n) = T(n - i) + i$. We might conclude that
$T(n) = T(n - (n - 1)) + (n - 1)$
$= T(1) + n - 1$
$=n-1$

Example) A slightly more complicated recurrence is
$T(n) = T(n - 1) + n$; $T(1) = 1$.
Expanding this recurrence a few steps, we get
$T(n) = T(n - 1) + n$
$= T(n - 2) + (n - 1) + n$
$= T(n - 3) + (n - 2) + (n - 1) + n.$
We should then observe that this recurrence appears to have a pattern that leads to
$T(n) = T(n - (n - 1)) + (n - (n - 2)) + \cdots + (n - 1) + n$
$= 1 + 2 + \cdots + (n - 1) + n = = n(n+1)/2$

Recursion

An algorithm is recursive if it calls itself to do part of its work.
In general, a recursive algorithm must have two parts: the base case, which handles a simple input that can be solved without resorting to a recursive call, and the recursive part which contains one or more recursive calls

Control flow

if ... then ... [else ...]
while ... do ...
repeat ... until ...
for ... do …
Indentation replaces braces

Method declaration

Algorithm method (arg [, arg...])
Input ...
Output …

Method call

method (arg [, arg...])

Return value

return expression

Expressions:

← Assignment
= Equality testing
n^2 Superscripts and other mathematical formatting allowed

END