

Forum:[Hack Forums](#)

Thread: [Recursive function to iterative function](#)

CarlosMarti123 - January 19, 2012, 9:49 pm

Hey everyone,

I have been trying to define a hyperoperation sequence in terms of recursion instead of iteration.

If you aren't familiar with hyperoperations, the [article](#) at Wikipedia can give you a quick idea about what this is all about. Basically, multiplication is repeated addition, exponentiation is repeated multiplication, etc. The next operation is called tetration, and it basically means repeated exponentiation. For instance A tetrated to B would mean A raised to itself B number of times. 2 tetrated to 3 (2^{2^3}) would be $2^{2^2^2}$. 2 tetrated to 4 (2^{2^4}) would be $2^{2^{2^2^2}}$ and so on. Addition is basically repeated succession, or +1. So $3 + 4$ can be defined as 3 plus 1 four times, or $3 + 1 + 1 + 1 + 1$. $2 + 5$ would be 2 plus 1 five times, or $2 + 1 + 1 + 1 + 1 + 1$.

Now, the algorithm for computing hyperoperations (or n-operations where n is an integer) can be defined [recursively](#) as follows:

For a hyperoperator n and two arguments a and b:

```
Code define hyper(n,a,b) as
if n = 0 then result = b + 1
else if n = 1 and b = 0 then [/b]result = a
else if n = 2 and b = 0 [b]then [/b]result = 0
else if n >= 3 and b = 0 [b]then [/b]result = 1
else result = hyper(n-1,a,hyper(n,a,b-1))
```

The problem is that functions that are defined recursively like this have many problems, and deeply-nested recursions can cause memory problems or run out of stack space. Is there anyone who can figure out how to turn this recursive function into an iterative one, e.g. using loops instead of calls?

I have made working formulas for addition:

```
Code result = A
loop(B)
{
result = result + 1
}
```

Which computes $A + B$, as well as for multiplication:

```
Code result = A
loop(B-1)
{
loop(A)
{
result = result + 1
}
}
```

Which computes $A \times B$, or A added to itself B number of times.

Note that all these functions are defined in terms of succession, or $+1$. So far, I haven't been able to extend this pattern for exponentiation, much less for tetration and other hyperoperations, so I have been searching for a way to convert the recursive function for computing a hyperoperation into an iterative function.

Any help with this? ?

E25EB8582FF05BF - January 20, 2012, 1:06 am

You already have flattened the multiplication. Make this a function. I don't know the language you are using. It should look like something like :

Code define multiply(A,B) as

```
result = A
loop(B-1)
{
  loop(A)
  {
    result = result + 1
  }
}
```

Now define a second function that looks like :

Code define raiseToPower(A,B) as

```
result = A
loop(B-1)
{
  result = multiply(result,A)
}
```

Then you tetration function :

Code define tetration(A,B) as

```
result = A
loop(B-1)
{
  result = raiseToPower(result,A)
}
```

CarlosMarti123 - March 10, 2012, 12:47 pm

Sorry for the late reply. Is there a way to merge these into a single function, a single function with an argument n for an n -level hyperoperation, rather than having to create a separate function for each value of n ?

MrZander - March 12, 2012, 12:40 pm

I've been attempting to do hyperexponentiation for a while in attempt to solve a problem on project euler.

So far, unsuccessfully.

All I know is that the numbers get VERY large VERY fast.

Keep that in mind when picking datatypes.

