Readings about DP and Recursion

Diego Linares - kiwiAipom

I. Everything about Dynamic Programming

II. Some general approach for solving recursive problems

Step One: Think about any input for which you know what your function should return.
Now suppose you have a task, related to a similar one. Keep calling that function to solve it: *I'll solve the problem if you give me this subproblem first*. Which is done by a call to the same function.
Example: With factorial, you only know that $0! = 1$ and $n! = n(n-1)!$. So the function that gives me factorial of n just needs the results of one that returns $(n-1)!$. This will keep going as long as we don't know whaat value to return.

```
factorial(n):
    if n = 0:
        return 1 // I know this, so I don't
            want my function to go any further
    else:
        return n*factorial(n-1) // just reuse
            the function
```

Step Two: They can do the same as loops, a simple `for` can be implemented as:

```
for(i, n):
    if i = n:
        return // Terminates
    // Do whatever needed
    for(i+1, n) // Next iteration
```

And for backwards:

```
rof(i, n):
    if i = n:
        return // Terminates
    rof(i+1, n) // Next iteration
    // Do whatever needed
```

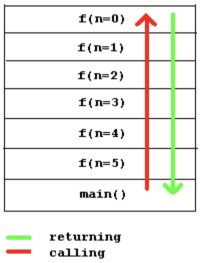
Since the function calls itself again until reaching a limit value and then starts returning.
Example: To print numbers backwards you may do this:

```
function(i, n):
    if i <= n:
        function(i+1, n)
        print(i)
```

Which for numbers from 1 to 5 would work like:

```
01|call function1 with i=1
02|    call function2 with i=2
03|        call function3 with i=3
04|            call function4 with i=4
05|                call function5 with i=5
06|                    call function6 with i=6
07|                        i breaks condition, no more calls
08|                    return to function5
09|                    print 5
10|                return to function4
11|                print 4
12|            return to function3
13|            print 3
14|        return to function2
15|        print 2
16|    return to function1
17|    print 1
18|return to main, done!
```

Step Three: There's a stack call which looks like this:



The memory of $f(3)$ for example, won't be freed until $f(2)$ is done. Serves the purpose of using an array, since the functions store variables and values.
Step Four: Be careful, in CP they are generally avoided since most can be done iteratively, andd they may exceed time and memory. Since every function is alloted a space at the moment it is called, might run into RTE. Use only $O(\lg n)$ and small $O(n)$ recursions.
Step Five: When there are overlapping branches in the recursion tree we store computed values (DP).

III. Dynamic programming 2

IV. Matrix

References

[1] *Attacking Recursions*, I Me and Myself. From: https://zobayer.blogspot.com/2009/12/cse-102-attacking-recursion.html