Divide y conquista

Introducción

Divide y conquista

Es partir un problema en problemas más pequeños, que no necesariamente sean instancias del mismo problema. Algunas formas en las que sucede esto es:

- Partiendo correctamente el problema en funciones auxiliares que sean conceptualmente más fáciles de crear y fáciles de utilizar.
- Resolviendo el problema para una familia de instancias y luego usar estas instancias para resolver las demás.
- Resolver el problema por casos.
- Resolver el problema de manera recursiva.

Algoritmos recursivos

Es una situación especial de la heurística de divide y conquista, en la cual ponemos a la instancia grande un problema en términos de instancias pequeñas del *mismo* problema.

Muchos lenguajes de programación permiten hacer cosas del siguiente estilo:

```
definir resolver(instancia grande):     decir qué hacer si estamos en el caso base     hacer cosas para obtener instancias pequeñas     resolver(instancias pequeñas)     hacer cosas para combinar soluciones de instancias pequeñas     regresar resultado
```

Aquí la función `resolver` se cita a sí misma, de manera muy parecida a como las sucesiones recursivas, como la de Fibonacci, están en términos de ellas mismas. Para que el algoritmo esté bien definido, la función debe decir que hacer con uno (o más) casos base. Veamos un par de ejemplos muy sencillos en Python:

- Uno en donde definimos la función «multiplicación por \(3\)» en términos de la función suma para los números naturales.
- Otro en donde definimos la función factorial.

```
def multi_3(n):
    if n==0:
        #Este es el "caso base" en donde decimos que multiplicar por cero da cero
        return 0
    else:
        #En esta linea es en la que hacemos recursión
        anterior=multi_3(n-1)
        return 3+anterior

print(multi_3(8))

def factorial(n):
    if n==0:
        return 1
    else:
        return n*(factorial(n-1))

print(factorial(100))
```

```
24
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852
```

Problema 15. Escribe un algoritmo recursivo que cree una lista de todas las permutaciones de \(n\) elementos.

Problema 16. Escribe un algoritmo recursivo que decida si una palabra es palíndroma o no, es decir, que diga si la palabra se lee igual al derecho que al revés.

al derecho que al revés.

Problema 15. Escribe un algoritmo recursivo que cree una lista de todas las permutaciones de \(n\) elementos.

Hagamos una lista de las permutaciones para poquitos elementos.

\(n=1\)

\(1\)

\(n=2\)

\(12, 21\)

\(n=3\)

\(123,132,213,231,312,321\)

\(n=4\)

?

Esbozo:

- Cuando tenemos un elemento, regresamos el elemento.
- Cuando tenemos \(n\) elementos, primero hacemos las permutaciones de \(n-1\) elementos y luego en cada una de ellas ponemos el \(n\)-ésimo elemento en cada posición.

```
def permutaciones(lista):
    if len(lista)==1:
        return([lista])
    else:
        anteriores=permutaciones(lista[:-1])
        ultimo=lista[-1]
        nuevas=[]
        for anterior in anteriores:
            for j in range(len(anterior)+1):
                nueva=anterior[:j]+[ultimo]+anterior[j:]
                nuevas.append(nueva)
        return(nuevas)

for j in permutaciones(['naranja','manzana','pera']):
    print(j)

for j in permutaciones([1,2,3,4]):
    print(''.join([str(d) for d in j]))
```

```
['pera', 'manzana', 'naranja']
['manzana', 'pera', 'naranja']
['manzana', 'naranja', 'pera']
['pera', 'naranja', 'manzana']
['naranja', 'pera', 'manzana']
['naranja', 'manzana', 'pera']
4321
3421
3241
3214
4231
2431
2341
2314
4213
2413
2143
2134
4312
3412
3142
3124
4132
1432
1342
1324
4123
1423
1243
1234
```

Problema 16. Escribe un algoritmo recursivo que decida si una palabra es palíndroma o no, es decir, que diga si la palabra se lee igual al derecho que al revés.

Para resolverlo en términos recursivos, debemos pensar en los casos base que tenemos. Pensaremos en que el caso base consiste de las palabras de o bien \(1\) caracter, o bien ningún caracter. En ambos casos, estas palabras son palíndromas.

Si tenemos una palabra más larga, es decir, con tres o más caracteres, una cosa que podemos hacer es comparar el primero con el último y ver si son iguales o no. Si no son iguales, entonces la palabra ya no fue palíndroma. Si sí son iguales, entonces el problema se reduce a que la palabra quitando el primero y el último sea palíndroma.

```
def palindroma(palabra):
    if len(palabra)==0 or len(palabra)==1:
        return True
    if palabra[0]!=palabra[-1]:
        return False
    else:
        return palindroma(palabra[1:-1])

print(palindroma('reconocer'))
print(palindroma('alegría'))
print(palindroma('a'))
print(palindroma(''))
print(palindroma('tassat'))
print(palindroma('tatsat'))
```

```
True
False
True
True
True
False
```

Sección 1

Sección 2

Sección 3

Tarea moral

Los siguientes problemas te ayudarán a practicar lo visto en esta entrada. Para resolverlos, necesitarás usar herramientas matemáticas, computacionales o ambas.

1. Problema
2. Problema
3. Problema
4. Problema
5. Problema

Anterior
Algoritmos voraces

Siguiente
Recursión y teorema maestro

Por Leonardo Ignacio Martínez Sandoval