Introdução à recorrência/recursividade em algoritmos: gerar todos os binários

Nesta seção apresentamos as ideias de como resolver um problema de forma recursiva, em particular ilustrando isso para resolver o seguinte problema: *como gerar todos os números binários com exatamente k bits (considerandos os "zeros à esquerda")?*.

Estrutura de um problema a ser resolvido recursivamente

Para que o problema seja natualmente resolvido por um algoritmo recursivo é preciso que ele seja facilmente quebrável em sub-problemas menores (denominamos esta técnica de resolução por **divisão e conquista**) e que depois as soluções dos problemas menores possam ser utilizadas para resolver o problema original.

O problema da geração dos binários com até *k* dígitos apresenta esta característica.

1. (**divisão**) Podemos quebrá-lo da seguinte forma: resolver para *k* dígitos pode ser feito resolvendo primeiro para *k-1* dígitos (nesse caso a quebra é "linear").
2. (**conquista**) Uma vez encontrada a solução `(l1, ..., lk)` para *k-1* dígitos, a solução para *k* dígitos é simplesmente `("0"+l1, ..., "0"+lk, "1"+l1, ..., "1"+lk)` (indicando por "+' o operador de concatenação).

A construção da solução recursiva para resolver o problema seria então aplicar a recorrência para cada um dos sub-problemas e depois usar os resoltados devolvidos para compor a **solução global**.

Resolvendo a geração ordenada de todos os números binário com até *k* dígitos

Desse modo, para tentar resolver um problema recursivamente, deve-se primeiro imaginar como quebrar o problema, então fazer um raciocínio de como reagrupar as soluções parciais para formar a solução global.

Para o problema considerado claramente a quebra poderia ser feita no número de dígitos. Se desejamos gerar os binários com *k* dígitos, supnhamos então que alguém resolveu o problema com *k-1* dígitos. Bem, se a solução para *k-1* dígitos é a lista `(0...0, 0...1, ..., 1...0, 1...1)`, então a solução global seria duplicar a lista dada, sendo que para cada uma das cópias adicionamos o dígito "0" à frente e na outra o dígito "1", ou seja, a solução seria a lista seguinte

```
00...0, 00...1, ..., 01...0, 01...1
10...0, 10...1, ..., 11...0, 11...1
```

Claro que um caso trivial, seria quando *k=1*, pois nesse caso *k-1=0* resultando em uma lista vazia. Portanto a solução global seria apenas os números "0" e "1" (eles concatenados com sequência vazia resultari neles próprios). E essa poderia ser a condição de terminação a recorrência: `se k=0, devolver <vazio>`.

Assim, vamos esquematizar o algoritmo refletindo precisamente essa ideia

```
gerar_binario (vet, n, k) {
  se (k==n) { // ja' gerou o ultimo bit =< imprima e volte
    imprima(vet, n);
    devolver; // funcao e' vazia
    }
  vet[k] = 0; // define o k com '0'
  gera_binario(vet,n,k+1); // indutivamente, gera os proximos
bits (de k+1 ate' n-1)
  vet[k] = 1; // altera o k
  gera_binario(vet,n,k+1); // idem
  }
```

Simule o algoritmo acima e veja que para *k=4* ele consegue construir a os 15 valores abaixo, na ordem correta.

Decimal	Binário	Decimal	Binário
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Exemplo: todos os binários com até 4 dígitos (com seu correspondente decimal à esquerda)

Leônidas de Oliveira Brandão
http://line.ime.usp.br

Alterações:
2020/08/20: acetos no formato, mais detalhes no gera_binario;
2020/08/15: novo formato, pequenas revisões;
2018/06/18: primeira versão