Trabalho Conversão de Bases

Gabriel Veiga

- Todas as bases numéricas estão representadas em base 10

Overview

Foi criada uma função em Python 3 que recebe como parâmetros:

1. Uma *string* correspondente à representação de um número inteiro positivo na base original (por exemplo “125”, “1111101”, “7D”);
2. A base em que o número foi representado (como um número). Se o valor do argumento anterior não for compatível com esta base, um erro será lançado;
3. A base em que se deseja representar o número.

E retorna dois valores,

1. Uma *string* com a representação do número na base desejada;
2. E a base dele (este valor pode ser facilmente ignorado).

Obs: Os números serão convertidos sempre que possível, então, por exemplo, uma entrada de `('0', 1, 10)`, ou seja, 0_1 para base 10, por mais que base 1 não faça sentido, irá retornar ‘0’, pois 0 representa o mesmo valor em todas as bases.

Da mesma forma, um número como 20_{10} poderá ser representado em qualquer base, sendo que 45_{10} não, pois, para bases maiores que o próprio valor, não existiria caracter no alfabeto romano para representá-lo. Ou seja, 35_{10} em base 36 ou maior é ‘Z’, e qualquer valor maior em base 37 ou maior é considerado não representável (em base 36 seria 10_{36}).

A função pode ser dividida em duas partes principais: primeiro, o valor de entrada é transformado em um valor numérico independente da base inicial, que, porque Python usa como padrão base 10, é o número em base 10 (em um nível abstrato, claro). Esta conversão é feita usando-se meramente a ideia de notação posicional (mais detalhes abaixo).

Em seguida, o número é convertido para a base desejada. O método para se fazer isso é o de divisões sucessivas.

Ambas as conversões acima são facilitadas por 10 ser a base numérica padrão em Python.

Outra parte importante do algoritmo, até então desconsiderada, é a seção que define duas funções que transformam caracteres (texto) em números e vice-versa. Isso é importante pois caracteres como “F” devem ter seu valor considerado como 15_{10}, e valores como 13_{10} devem ser mostrados como “D” quando possível.

Detalhes

Primeiramente, define-se as duas funções para conversão de caractere de texto para número e vice-versa:

```
def __get_decimal_value(charnumber):

   charnumber = charnumber[0].upper()

   if charnumber >= '0' and charnumber <= '9':
       return int(charnumber)
   else:
       if charnumber >= 'A' and charnumber <= 'Z':
           return 10 + ord(charnumber) - 65
       else:
           raise ValueError('Algarismo "{}" nao
reconhecido'.format(charnumber))

def __get_symbol(number):

   if number >= 0 and number <= 9:
       return str(number)
   else:
       if number <= 35:
           return chr(number - 10 + 65)
       else:
           raise ValueError('Numero {} nao representavel'.format(number))
```

Basicamente o que elas fazem é, quando possível, usar as funções `int()` e `str()` do Python para converter valores numéricos em base 10 diretamente ('6' ↔ 6) e, quando não possível, calcular os seus valores com base na tabela *ascii*, de forma que 'A' ↔ 10. A linha "`charnumber = charnumber[0].upper()`" garante que 'A' e 'a' representam a mesma coisa.

Em seguida, calculamos o valor do número em base 10 usando a função `__get_decimal_value`:

• • •

```
decval = 0
origbase_exp = len(val) - 1

for char in val:
   # decimal_value corresponde a um unico digito
   decimal_value = __get_decimal_value(char)

   if decimal_value < origbase:
       decval += decimal_value * ( origbase ** origbase_exp )
       origbase_exp -= 1
   else:
       raise ValueError('Algarismo {} nao faz sentido para origbase
{}'.format(char, origbase))
```

Aqui, inicializamos duas variáveis, *decval* e *origbase_exp*. A primeira conterá o valor do nosso número em base 10 (praticamente) e a segunda nos auxiliará a calcular o valor final da primeira.

O procedimento é bem simples: para cada caractere do número original, calcula-se o seu valor numérico (`__get_decimal_value`), multiplica-se pelo seu peso ($base^{posição}$) e soma-se o seu valor no valor de *decval*.

Isso é equivalente a $x = n_k \cdot b^k + n_{k-1} \cdot b^{k-1} + \ldots + n_0 \cdot b^0$, onde **n** é o conjunto dos números usados para a representação (i.e. *val*), **b** é a base do número original (i.e. *origbase*), **k** é a posição do número, contando da direita, começando em zero, para a esquerda (neste caso o uso do *for* deixou pouco confuso, mas pode-se dizer que este é o valor de *origbase_exp*), e **x** é o valor total convertido (i.e. *decval*).

Depois, convertemos o nosso valor em base 10 para o valor na base especificada.

```
if newbase == 10:
   return str(decval), 10
```

Se a base especificada for 10, não há mais cálculos a se fazer.

• • •

```
numarr = []

aux = decval
while aux >= newbase:
   numarr.append(aux % newbase)
   aux //= newbase

numarr.append(aux)

returnstr = ''.join([__get_symbol(n) for n in numarr[::-1]])

return returnstr, newbase
```

Se não, cria-se uma *list* (sim, o nome engana) para guardar os caracteres do novo número e começa-se o processo de divisão:

1. Cria-se uma variável auxiliar, inicialmente contendo o valor decimal calculado anteriormente;
2. Enquanto o valor desta variável for maior ou igual à nova base, ou seja, enquanto $\frac{aux}{newbase} \geq 1$, divide-se aux por newbase, guardando o resto da operação no final da list;
3. Enfim, quando aux for menor que newbase, seu valor é adicionado no final da *list* também.

Para entender como isso funciona, observe o que acontece com os restos das divisões (números azuis) à medida que o número 3864_{10} é dividido por 8_{10} (lembrando que $8^0 = 1$):

$$3864 = 7 \cdot 8^3 + 4 \cdot 8^2 + 3 \cdot 8^1 + 0 \cdot 8^0 = 7430_8$$

$$7 \cdot 8^2 + 4 \cdot 8^1 + 3 \cdot 8^0 = \frac{7430_8}{8} = 483,\ resto\ 0$$

$$7 \cdot 8^1 + 4 \cdot 8^0 = \frac{7430_8}{8^2} = 60,\ resto\ 3$$

$$7 \cdot 8^0 = \frac{7430_8}{8^3} = 7,\ resto\ 4.\ 7 < 8.$$

$$0,\ 3,\ 4,\ 7 \rightarrow 7430$$

Efetivamente o que acontece é que, a cada passo, quando dividimos o número que temos pela base, obtemos no resto o termo $n \cdot 8^0$, ou seja, o último dígito do número na base 8. Se realizado várias vezes, todos os dígitos do número podem ser obtidos, só que na ordem inversa.

```
returnstr = ''.join([__get_symbol(n) for n in numarr[::-1]])

return returnstr, newbase
```

Uma vez que isso tenha sido feito, percorre-se a *list* de números recém calculados ao contrário, obtém-se o símbolo adequado para cada um e concatena-se todos em uma *string*, que é retornada.