Computação 1, 2020.1
Lista 8

Para entrega até 18/2 às 10:00

Submeta suas soluções colocando os arquivos correspondentes na sua pasta do Google Drive

Lembre-se de escolher bons nomes para suas funções e variáveis, de documentar seu código com *docstrings* (documentação de função) e comentários onde for apropriado, e de fazer testes para suas funções (na documentação usando o módulo `doctest` ou em funções dedicadas).

Questão 1 (Sorteios)**.** Faça uma função que receba um natural `N` e retorne a quantidade de sorteios necessários para que, sorteando números aleatoriamente de `0` a `N`, encontre-se um número repetido pela primeira vez. Idealmente, sua função deve ter *boa performance* (i.e., funcionar em "até alguns segundos" em média) para `N` com até 10 algarismos.

Questão 2 ("Invertendo" dicionários)**.**

a. Faça uma função que receba um dicionário `D` e retorne o dicionário "inverso", i.e., um dicionário `D'` tal que o par `c:v` está associado em `D'` sse o par `v:c` está associado em `D`. Quais propriedades o dicionário de entrada tem que satisfazer para a sua função funcionar sem erros?

b. Faça uma função que receba um dicionário `D` e retorne o dicionário "inverso" um pouco melhor: um dicionário `D'` tal que o par `c:v` está associado em `D'` sse `v` é exatamente o **conjunto** de chaves de `D` associadas ao conteúdo `c`. Quais propriedades o dicionário de entrada tem que satisfazer para a sua função funcionar sem erros?

Questão 3 (Polinômios)**.** Nesta questão, vamos representar polinômios usando dicionários. O polinômio na variável x

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

será representado por um dicionário cujas chaves são os graus $d \leq n$ e os valores correspondentes são os coeficientes a_d. (Note que, como os coeficientes podem ser nulos, qualquer polinômio pode ser representado por vários dicionários diferentes.)

Para cada item abaixo, faça uma função...

a. ... para avaliar um polinômio dado em um ponto x dado.

b. ... para retornar a *função polinomial* correspondente a um polinômio dado (i.e., a função que recebe x e retorna o valor do polinômio no ponto x).

c. ... para somar uma quantidade finita de polinômios (dados em uma lista).

d. ... para multiplicar um polinômio por um escalar.

e. ... para multiplicar uma quantidade finita de polinômios (dados em uma lista)

f. ... para derivar um polinômio.

g. ... para *antiderivar* um polinômio (i.e., dado um polinômio p, deve-se retornar um polinômio q qualquer cuja derivada seja p).

h. ... para encontrar uma raiz de um polinômio dado próxima de um ponto x dado (assumindo que tal raiz exista).

Desafio (opcional)

Questão 4 (Uma função hash)**.** Como vimos, o `python` usa funções *hash* na implementação de conjuntos e dicionários. Intuitivamente, podemos imaginar que o hash de um objeto indica o *endereço* em que aquele objeto ocupa (ou deveria ocupar) no conjunto ou dicionário em questão. (A questão 5 traz mais detalhes.)

Neste contexto, uma função hash é uma função que dado um valor qualquer retorna um número inteiro, com a seguinte propriedade adicional: para dois objetos `x`, `y`, se `x` e `y` têm o mesmo *valor* (i.e., se `x == y` for `True`) então a função hash retorna o mesmo inteiro para ambos. Essa propriedade garante, por exemplo, que recuperar um valor em um dicionário usando sua chave funcione como deveria.

Nesta questão, vamos usar a seguinte função hash:

```
def hash_simples(x, M = 10000):
    if isinstance(x,(int,complex,float)):
        return int(x) % M # comentários 1 e 2 abaixo
    elif isinstance(x,(bool,str)) or x == None:
        return id(x) % M # comentário 3 abaixo
    # se chegamos aqui é porque x tem um tipo "composto";
    # vamos calcular seu hash "compondo" os hashes de seus elementos
    res = id(type(x)) % M # comentário 4 abaixo
    for i,z in enumerate([hash_simples(y) for y in x]):
        res ^= (2**(i+1))*z  # comentários 5 e 6 abaixo
    return res % M
```

Comentários sobre a função `hash_simples`:

1. Como (na maioria das implementações de `python`) podemos ter `(x is y)` falso mesmo com `(x == y)` verdadeiro para `x, y` inteiros/complexos/float, não devemos usar a função `id` aqui.

2. Trabalhamos (mod M) para que os valores de hash sejam relativamente pequenos e evitar problemas de memória mais adiante.

3. Objetos de tipo `bool`, `str` e `NoneType` são "únicos em memória", i.e., para estes tipos vale que `(x == y)` implica `(x is y)`, então podemos usar `id` para calcular `hash_simples`.

4. Isto serve para que `(1,2,3)` tenha hash diferente de `[1,2,3]`, por exemplo.

5. A operação `^` é chamada *bitwise XOR*: dados inteiros x e y, o inteiro $x \hat{} y$ é aquele cuja representação binária tem um bit 0 em uma posição sse x e y têm bits iguais em suas representações, nessa mesma posição. Assim, $10 \hat{} 14 = 4$ pois $10 = (1010)_2$, $14 = (1110)_2$ e $(0100)_2 = 4$.

6. Multiplicamos cada `z` por uma potência de 2 diferente para que `[1,2,3]` tenha hash diferente de `[2,1,3]`, por exemplo.

Quando objetos `x` e `y` tais que `x != y` têm o mesmo valor de hash, isso é chamado de *colisão*. No caso da nossa função `hash_simples`, como estamos usando apenas `M` valores possíveis de hash, teremos muitas colisões.

a. Encontre algum `x` que esteja em colisão com a string cujo valor é o seu nome.

b. Faça um esquema que ilustre a execução da chamada `hash_simples([1,2,3])`, indicando os valores de todas as variáveis e chamadas recursivas ao longo do processo.

c. Explique como pode acontecer do valor de hash de algum objeto mudar ao longo da execução de um programa. Por que isso é um problema para o uso dessa função hash para "localizar" objetos em conjuntos e dicionários?

d. Caracterize os objetos cujos valores de hash nunca mudam ao longo da execução de qualquer programa (não importando os comandos executados no programa!).

Questão 5 (Implementando dicionários simples)**.** Nesta questão vamos usar a função `hash_simples` do item anterior.

Uma forma (simples) de implementar um dicionário é usar uma lista onde cada elemento é `None`, ou uma tupla `(chave,valor)`, ou a string `''deletado''`. A intenção é que o par `(chave,valor)` ocupe o índice `hash_simples(chave)` da lista; entretanto, por causa de colisões, isso pode não ser verdade.

Nos itens abaixo, "dicionário" sempre quer dizer uma lista como acabamos de descrever.

a. Faça uma função que receba um inteiro `N` (com valor padrão `0`) e retorne um dicionário de tamanho `N` onde todas as posições são ocupadas por `None`.

b. Faça uma função que receba um dicionário `D` e um inteiro `N` e aumente o tamanho de `D` em `N` unidades (`D` deve ser modificado). Todas as novas posições devem ser ocupadas por `None`.

c. Faça uma função que receba um dicionário `D`, uma chave `c` e um valor `v` e "adicione ou atualize a associação de `c` a `v` em `D`": o par `(c,v)` deve ser inserido em `D` na primeira posição a partir de `hash_simples(c)` que satisfaça alguma das seguintes condições:

1. o elemento nessa posição é `None`; ou

2. o elemento nessa posição é `''deletado''`; ou

3. o elemento nessa posição é da forma `(c,v')` para algum `v'`.

I.e., em cada caso, o elemento que ocupava essa posição em `D` deve ser substituído por `(c,v)`. O dicionário `D` deve ser aumentado caso necessário.

d. Faça uma função que receba um dicionário `D` e uma chave `c` e retorne o valor associado a `c` em `D`, caso exista. Lembre-se que, por causa de colisões, o valor associado a `c` pode não estar na posição `hash_simples(c)`. Caso `c` não esteja associado a nenhum valor em `D`, em que momento ao longo da sua "busca" você pode concluir isso com certeza?

e. Faça uma função que receba um dicionário `D` e uma chave `c` e remova a associação de `c` em `D` (caso exista), substituindo o elemento correspondente de `D` pela string `''deletado''`. (Por que não simplesmente trocar por `None`?)