26 Aula 26: 12/11/2019

26.1 Hoje

Aula de exercícios.

O exercício 1 é uma versão dos problemas vistos na aula 20

O exercício 2 está relacionado com o Método de Monte Carlo. Esse foi o tópico das aula 24 e aula 25

26.2 Arquivos

As soluções estão no diretório `py`:

- `2sum`: exercício 1
- `monte_carlo`: exercício 2

26.3 Execícios 1

Considere o seguinte problema

2SUM: dada uma lista de números inteiros determinar o número de pares que somam zero.

(a) Escreva uma função que resolve o problema e consome tempo $O(n^2)$.

(b) Escreva uma função que resolve o problema e consome tempo $O(n \lg n)$.

(c) Escreva uma função que resolve o problema e consome tempo **esperado** $O(n)$.

Objetivos

Este exercício é sobre análise de algoritmos. Usamos esse problema em um laboratório de ideias.

A *primeira solução* testa todas as possibiliades e consome tempo $O(n^2)$.

A *segunda solução* combina ordenação e busca binária e consome tempo $O(n \lg n)$.

Por fim, a *terceira e última solução* consome tempo **esperado** $O(n)$.

Com isso deveriamos notar que ordenação, busca binária e `set` e `dict` são ferramentas valiosas.

Solução (a)

```
def soma_zero_n2(v):
    '''(list) -> int
    Recebe uma lista `v` de números inteiros e retorna o número
    de pares que somam zero.

    Consum de tempo da função é O(n^2).
    '''
    cont = 0
    n = len(v)
    for i in range(n):
        for j in range(i+1,n):
            if v[i] + v[j] == 0:
                # print(v[i], v[j])
                cont += 1
    return cont
```

Solução (b)

Em uma das versões usamos a busca binária da biblioteca e na outra implementamos a busca binária.

Nota: No dia a dia, sempre que possível, devemos usar funções de bibliotecas.

```
def soma_zero_nlgn(v):
    '''(list) -> int
    Recebe uma lista `v` de números inteiros e retorna o número
    de pares que somam zero.

    Consumo de tempo da função é O(n lg n).

    A função altera a lista v. Para não alterarmos bastava
    fazermos v_sorted = sorted(v)
    '''
    cont = 0
    v.sort()
    n = len(v)
    for i in range(n):
        j = index_busca_binaria(v, -v[i], i+1, n) # consumo de tempo O(lg n)
        # j = index(v, -v[i], i+1, n) # alternativamente
        if j != None: cont += 1
    return cont


def index_busca_binaria(a, x, lo, hi):
    '''(list, item, int, int) -> int ou None

    Recebe uma lista ordenada a com n itens, um item x e dois inteiros lo e hi.
    Retorna um índice i em range(lo,hi) tal que a[i] <= x < a[i+1].
    Para essa afirmação fazer sentido suponha aqui que
        * a[lo-1] é menos infinito e
        * a[hi  ] é mais  infinito.
    Pré-condição: suponha que os itens em a são distintos.
    '''
    while lo < hi:
        mid = (lo + hi) // 2 #
        if   a[mid] < x: lo = mid + 1
        elif a[mid] > x: hi = mid
        else: return mid
    return None

# Mais especificamente, usaremos uma adaptação de `index` da página de `bisect`
def index(a, x, lo, hi):
    '''(list, item, int, int) -> int ou None
    >>> from bisect import bisect_left
    >>> lista = [1,3,5,7,9,11]
    >>> help(bisect_left)
```

```
    >>> bisect_left(lista,2,0,len(lista))
    1
    >>> bisect_left(lista,13,0,len(lista))
    6
    >>> bisect_left(lista,3,0,len(lista))
    1
    >>> bisect_left(lista,4,0,len(lista))
    2
    >>> bisect_left(lista,5,0,len(lista))
    2
    >>> bisect_left(lista,0,0,len(lista))
    0
    >>>
    '''
    i = bisect_left(a, x, lo, hi)
    if i != len(a) and a[i] == x:
        return i
    return None # raise ValueError
```

Solução (c)

```
def soma_zero_n(v):
    '''(list) -> int
    Recebe uma lista `v` de números inteiros e retorna o número
    de pares que somam zero.

    Consumo de tempo esperado da função é O(n).
    '''
    cont = 0
    numeros = set() # conjunto de valores
    n = len(v)
    for i in range(n):
        if -v[i] in numeros: # consumo de tempo esperado O(1)
            cont += 1
        numeros.add(v[i])    # consumo de tempo esperado O(1)
    return cont
```

Experimentos

```
% python doublingTest.py -q 16Kints.txt
main(): lendo lista de inteiros...
main(): lista com 16000 ints lida

      n       tempo  pares
      8     0.000s      0
     16     0.000s      1
     32     0.000s      1
     64     0.000s      1
```

```
   128     0.001s     1
   256     0.003s     1
   512     0.012s     1
  1024     0.046s     1
  2048     0.176s     2
  4096     0.693s     4
  8192     2.777s     19
 16000    10.541s     66

% python doublingTest.py -o 16Kints.txt
main(): lendo lista de inteiros...
main(): lista com 16000 ints lida

      n      tempo  pares
      8     0.000s     0
     16     0.000s     1
     32     0.000s     1
     64     0.000s     1
    128     0.000s     1
    256     0.000s     1
    512     0.001s     1
   1024     0.002s     1
   2048     0.004s     2
   4096     0.009s     4
   8192     0.018s     19
  16000     0.037s     66

% python doublingTest.py -l 16Kints.txt
main(): lendo lista de inteiros...
main(): lista com 16000 ints lida

      n      tempo  pares
      8     0.000s     0
     16     0.000s     1
     32     0.000s     1
     64     0.000s     1
    128     0.000s     1
    256     0.000s     1
    512     0.000s     1
   1024     0.000s     1
   2048     0.000s     2
   4096     0.001s     4
   8192     0.002s     19
  16000     0.004s     66

% python doublingTest.py -l 1Mints.txt
main(): lendo lista de inteiros...
main(): lista com 1000000 ints lida

      n      tempo  pares
```

```
      8     0.000s     0
     16     0.000s     1
     32     0.000s     1
     64     0.000s     1
    128     0.000s     1
    256     0.000s     1
    512     0.000s     1
   1024     0.000s     1
   2048     0.000s     2
   4096     0.001s     4
   8192     0.002s     19
  16384     0.003s     71
  32768     0.008s     290
  65536     0.017s     1102
 131072     0.038s     4350
 262144     0.087s     17267
 524288     0.201s     68673
1000000     0.431s     249838

% python doublingTest.py -o 1Mints.txt
main(): lendo lista de inteiros...
main(): lista com 1000000 ints lida

      n      tempo  pares
      8     0.000s     0
     16     0.000s     1
     32     0.000s     1
     64     0.000s     1
    128     0.000s     1
    256     0.000s     1
    512     0.001s     1
   1024     0.002s     1
   2048     0.003s     2
   4096     0.008s     4
   8192     0.018s     19
  16384     0.039s     71
  32768     0.079s     290
  65536     0.168s     1102
 131072     0.370s     4350
 262144     0.802s     17267
 524288     1.760s     68673
1000000     3.657s     249838

% python doublingTest.py -q 1Mints.txt
main(): lendo lista de inteiros...
main(): lista com 1000000 ints lida

      n      tempo  pares
      8     0.000s     0
     16     0.000s     1
```

```
   32     0.000s     1
   64     0.000s     1
  128     0.001s     1
  256     0.003s     1
  512     0.012s     1
 1024     0.047s     1
 2048     0.188s     2
 4096     0.750s     4
 8192     2.967s     19
16384    12.107s     71
32768    49.683s     290
65536   200.434s     1102
Abortei pois iria cansar de esperar...
```

26.4 Exercício 2

Prova 1 de MAE0212, 2019

Um professor(a) dá um teste rápido, constante de 36 questões do tipo certo-errado. Para testar a hípótese de um/uma estudante estar adivinhando a resposta, é adotada a seguinte regra de decisão:

> "*Se 22 ou mais questões estiverem corretas, o/a estudante não está adivinhando*"

Qual é a probabilidade de rejeitarmos a hipótese, sendo que na verdade é verdadeira?

Ideias

Na últimas aulas vimos um certo arcabouço para esse tipo de problema. Aqui, mais uma vez, veremos esse arcabouço.

Modelagem

Utilizaremos uma simulação em que uma lista `gabarito` com 36 contedo `True` ou `False` é escolhida uniformemente ao acaso

```
self.gabarito = [random.choice([True, False]) for i in range(36)]
```

Poderíamos simplemente supor que todas as respostas eram `True` e prosseguir com a simulação, mas talvez alguns e algumas se sintam melhor desta forma.

Em cada experimento, modelamos uma pessoa chutando respostas uniformente ao acaso com o seguinte trecho de código:

```
chute = random.choice([True, False])
```

O experimento é considerado sucesso se os acertos forem menor que 22. Isso significa que a hipótese é boa.

Os experimentos abaixo mostraram que a probabilidade de chutando alguém acertar mais que 22 questões é menor `0.13`.

Supomos que na prova de MAE0212 as alunas e alunos tiveram que estimar esse valor olhando para a gaussiana de media $\mu = 36 \times 0.5 = 18$ e variância $\sigma = \sqrt{36 \times 0.5 \times 0.5} = 3$.

Foi isso?

Cliente

```
def main():
    n_questoes = int(input("Digite o no de questões: "))
    limiar     = int(input("Digite o limiar para chute: "))
    t          = int(input("Digite o no de experimentos: "))
    adilson = Cientista(n_questoes, limiar, t) # nome do prof
    print("Probabilidade da/do estudante estar chutando", adilson.mean())
```

Cientista

Essa classe teve nomes diferentes dependendo do problema, mas o esquema foi sempre o mesmo. Alguns nomes foram `Aniversario`, `Colecionador`, `Area`,... Achamos que simulação é uma ferramenta importante e que deveria ser valorizada.

```
import random

class Cientista:
    def __init__(self, no_questoes, limiar, t):
        self.no_questoes = no_questoes
        self.limiar = limiar
        # o gabarito poderia ser toda resposta True...
        self.gabarito = [random.choice([True,False]) for i in range(no_questoes)]
        # sucesso significa que chutando a/o estudante acertou
        # menos que limiar questoes
        sucesso = 0
        for i in range(t):
            sucesso += self.experimento()
        self.p = sucesso/t

    def mean(self):
        return self.p

    def experimento(self):
        n        = self.no_questoes
        limiar   = self.limiar
        gabarito = self.gabarito
        acertos = 0
        for i in range(n):
            chute = random.choice([True,False])
            if chute == gabarito[i]:
                acertos += 1
        if acertos < limiar: return 1
        return 0
```

Experimentos

Os experimentos mostram que a probabilidade de alguém acertar mais que 22 questões chutando é menor que 0.13.

```
% python main.py
Digite o no de questões: 36
Digite o limiar para chute: 22
Digite o no de experimentos: 100
Probabilidade da/do estudante estar chutando 0.87

% python main.py
Digite o no de questões: 36
Digite o limiar para chute: 22
```

```
Digite o no de experimentos: 1000
Probabilidade da/do estudante estar chutando 0.871

% python main.py
Digite o no de questões: 36
Digite o limiar para chute: 22
Digite o no de experimentos: 10000
Probabilidade da/do estudante estar chutando 0.88


% python main.py
Digite o no de questões: 36
Digite o limiar para chute: 22
Digite o no de experimentos: 100000
Probabilidade da/do estudante estar chutando 0.88046

% python main.py
Digite o no de questões: 36
Digite o limiar para chute: 22
Digite o no de experimentos: 1000000
Probabilidade da/do estudante estar chutando 0.878952
```