24 Aula 24: 04/NOV/2019

24.1 Aulas anteriores

Nas aulas anteriores conversamos sobre:

- **aula 21**: Programação dinâmica: *Longest Commom Subsequence* (LCS)
- **aula 22**: vetor de sufixos, ordenação, fatias x vistas: *Longest Repeated Substring* (LRS)
- **aula 23**: `set` do Python: `set` é um `dict` com apenas chaves e sem valores associados; consumo de tempo esperado `O(1)` para maioria das operações

24.2 Hoje

Trabalharemos com o Método de Monte Carlo e um certo arcabouço para simulações computacionais. Ver as páginas Monte Carlo method na Wikipedia e Monte Carlo Simulation de IntroCS de Princeton. Se possível falaremos de

Trataremos do

- paradoxo do aniversário e
- colecionador de figurinhas

24.3 Arquivos

Todos os diretórios em py têm um arquivo `main.py` que o cliente da classe principal:

- `aniversario` (`aniversario.py`) para o paradoxo do aniversario
- `colecionador` (`colecionado.py`) para o colecionador de figurinhas

Todos os diretórios tem um arquivo `config.py` com constantes para os programas. As constantes poderiam ser parâmetros. As classe são parecidas. No `main.py` está implementado *adaptative plotting*.

O diretorio `imgs` tem uma imagens geradas pelo programa do paradoxo do aniversário.

24.4 Método de Monte Carlo

O método de **Monte Carlo**, ou experimentos de Monte Carlo, são uma ampla classe de algoritmos computacionais que dependem de amostragem aleatória repetida para obter resultados numéricos. Vocês usaram esse método para estimas o limiar de percolação de um sistema no EPXY. Agora applicaremos em problemas em que é possível determinar a resposta analiticamente; diferentemente do que ocorre com percolação.

Da página de Princeton:

> Em 1953, Enrico Fermi, John Pasta e Stanslaw Ulam criaram o primeiro "experimento computacional" para estudar uma estrutura atômica vibratória. O sistema não linear não podia ser analisado pela matemática clássica.

Simulação: método analítico que imita um sistema físico.

Simulação de Monte Carlo = usa valores gerados aleatoriamente para variáveis incertas. Nomeado em referência ao famoso cassino em Mônaco.

Em cada etapa da evolução do cálculo, repita várias vezes para gerar uma variedade de cenários possíveis e resultados médios.

O método é usado quando outras técnicas falham. Normalmente, a precisão é proporcional à raiz quadrada do número de repetições. Tais técnicas são amplamente aplicadas em vários domínios, incluindo:

- projeto de reatores nucleares,
- prever a evolução das estrelas,
- prever o mercado de ações etc.

24.5 Lei dos grandes números

A **lei dos grandes números** diz que a média aritmética dos resultados da realização da mesma experiência repetidas vezes tende a se aproximar do valor esperado à medida que mais tentativas se sucederem. Em outras palavras, quanto mais tentativas são realizadas, maior a probabilidade da média aritmética dos resultados observados se aproximarem da probabilidade real.

24.6 Aleatoriedade em Python

Em `Python` temos a classe `random` com vários métodos e funções para gerarmos objetos aleatórios. Veja https://docs.python.org/3/library/random.html. Entre esses métodos e funções temos alguns que já usamos:

- `random.seed(a=None, version=2)`: inicializa o gerador de números aleatórios
- `random.randrange(start, stop[, step])`: retorna um valor escolhido uniformemente ao acaso em `range(start, stop[,step])`
- `random.choice(lst)`: retorna um item escolhido uniformemente ao acaso da lista `lst`.
- `random.shuffle(lst[, random])`: embaralha *in place* os itens na lista `lst`.
- `random.sample(populacao, k)`: retorna uma lista com `k` itens da `polulacao` que é uma `list`or `set`

```
Python 3.7.3 (default, Mar 27 2019, 22:11:17)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> random.seed("oi")
>>> random.randrange(27)
7
>>> random.seed("oi")
>>> random.randrange(27)
7
>>> random.seed("Eduardor")
>>> random.randrange(27)
14
>>> lst = ["joão", "pedro", "juliana", "giovana", "julia"]
>>> random.choice(lst)
'giovana'
>>> random.choice(lst)
'julia'
>>> random.choice(lst)
'julia'
>>> random.choice(lst)
'pedro'
>>> lst
['joão', 'pedro', 'juliana', 'giovana', 'julia']
>>> random.shuffle(lst)
>>> lst
['julia', 'juliana', 'pedro', 'giovana', 'joão']
>>> random.shuffle(lst)
>>> lst
['giovana', 'joão', 'pedro', 'juliana', 'julia']
>>> random.sample(lst,2)
['giovana', 'julia']
>>> random.sample(lst,2)
['juliana', 'pedro']
>>> random.sample(lst,2)
['joão', 'julia']
>>>
```

24.7 Problema paradoxo do aniversário

Problema

Será que nessa sala temos 2 pessoas que fazem aniversário no mesmo dia? Qual a chance disso acontecer em uma sala com 20 alunos e alunas? E com 40 alunos e alunas?

Como a gente pode usar computação para explorar esse problema?

Suponha que pessoas entram em uma sala inicialmente vazia até que tenhamos duas pessoas que façam aniversário no mesmo dia.

Escreva um programa para estimar quantas pessoas entrarão na sala até que isso ocorra.

Probabilidade teórica

Para calcular aproximadamente a probabilidade de que em uma sala com n pessoas, pelo menos duas possuam o mesmo aniversário, desprezamos variações na distribuição, tais como anos bissextos, gêmeos, variações sazonais ou semanais, e suporemos que 365 possíveis dias de aniversários são todos igualmente prováveis.

Distribuições de aniversários na realidade não são uniformes uma vez que as datas não são equiprováveis.

É mais fácil calcular a probabilidade $\overline{p}(n)$ de que todos os n aniversários sejam diferentes. Se $n > 365$, pelo *Princípio da Casa dos Pombos* esta probabilidade é 0. Por outro lado, se $n \leq 365$, ela é dada por

$$\begin{aligned}\overline{p}(n) &= 1 \times (1 - 1/365) \times (1 - 2/365) \cdots (1 - (n-1)/365) \\ &= \frac{365 \times 364 \cdots (365 - n + 1)}{365^n} \\ &= \frac{365!}{365^n (365 - n)!}.\end{aligned}$$

n	$p(n)$
8	0.07
10	0.11
12	0.16
16	0.28
18	0.34
20	0.41
22	0.47
24	0.53
30	0.70
40	0.89

Uma curiosidade que talvez seja útil mais adiante é que funções injetoras são raras. Se considerarmos todas as funções de `range(23)` a `range(365)` essa contas mostram que mais de metade delas são não injetoras.

Modelagem

Vamos partir para programação.

Por simplicidade suporemos que cada pessoa é igualmente provável de ter nascido em qualquer um dentre 365 dias possíveis (ignore 29 de fevereiro).

Gere um número aleatório entre 0 e 364 para cada pessoa e verifique se a outra pessoa na sala e para assim que o mesmo valor for gerado duas vezes.

Execute esse experimento t vezes. (t = número de trials)

24.8 Arcabouço

O código abaixo é como a classe `Stats` do EP sobre percolação.

O código abaixo utiliza a classe nativa `set`. Podemos usar `list`, mas como sabemos, operações com `set` e `dict` são mais eficiente que operações sobre `list`.

```
class Aniversario:
    #------------------------------------------
    def __init__(self, n, t = T):
        '''(int, int) -> None

        Recebe o número n de datas sorteadas e o número t
        de experimentos (trials) e calcula a probabilidade
        de selecionando n datas uniformemente ao acaso tenhamos
        duas datas iguais.
        '''
        self.n = n
        self.t = t
        sucessos = 0
        for i in range(t):
            sucessos += self.experimento()
        self.p = sucessos/t # guarda a probabilidade

    #------------------------------------------
    def mean(self):
        return self.p

    #-----------------------------------------
    def experimento(self):
        n = self.n
        aniversarios = set()
        for i in range(n):
            data = random.randrange(DATAS) # valor entre 0 e DATAS-1
            if data in aniversarios:
               return True
            aniversarios |= {data} # união de conjuntos
        return False
```

24.9 Cliente

O programa cliente dessa classe para o nosso experimento é o seguinte. Essa é uma versão simplificada.

```
from aniversario import Aniversario

# parâmetros: GAP, ERR, X0, Y0, X1, Y1, T
from config import *

#-------------------------------------------------------
def main():
    n = int(input("Digite o número de datas: "))
    t = int(input("Digite o número de experimentos: "))

    # realize os t experimentos
    ani = Aniversario(n, t)

    # média aritmética
    print("probabilidade =", ani.mean(), "(%f)"%prob(n))


#-----------------------------------------
def prob(n):
    '''
    Retorna a probabilidade teorica.

    import math
    num = math.factorial(DATAS)
    den = math.factorial(DATAS-n)
    prob_c = num/(DATAS**n * den)
    '''
    prob_c = 1
    for i in range(n):
        prob_c *= (1 - i/DATAS)
    return 1 - prob_c
```

24.10 Plot

A seguir há duas versões de plotagem. Uma em que os pontos são plotados em um passo de `GAP` e outro em que a plotagem é adaptativa.

Plot de passo fixo

```python
# parâmetros: GAP, ERR, X0, Y0, X1, Y1, T
from config import *

import matplotlib.pyplot as plt

#-------------------------------------------------------------
def grafico():
    '''
    Referência:
    https://panda.ime.usp.br/algoritmos/static/algoritmos/10-matplotlib.html
    '''
    plt.title("MAC0122 Paradoxo do Aniversário")
    plt.xlabel("número de pessoas")
    plt.ylabel("probabilidade")
    x = []
    y = []
    x0, y0 = X0, Y0
    for n in range(X0, X1, GAP):
        # realize os no_exp experimentos
        ani = Aniversario(n, T)
        x1 = n
        y1 = ani.mean()
        x.append(x1)
        y.append(y1)
        plt.plot([x0,x1],[y0,y1], 'b-') #mlines.Line2D([x0,y0], [x1,y1])
        x0, y0 = x1, y1
    plt.plot(x, y, 'go') # green bolinha
    plt.show()
```

Plot adaptativo

Usado quando os experimentos consomem muito tempo e devemos escolher os valores a serem testados de maneira mais parsimoniosa. Ideia recursiva.

```python
import matplotlib.pyplot as plt

#-------------------------------------------------------------
def adaptative_plot():
    plt.title("MAC0122 Paradoxo do Aniversário")
    plt.xlabel("número de pessoas")
    plt.ylabel("probabilidade")

    # desenhe o ponto inicial e final
    plt.plot([X0, X1], [Y0, Y1], 'go') # green bolinha

    # desenhe recursivamente a curva entre eles
    curva(X0, Y0, X1, Y1)

    # mostre o gráfico
    plt.show()

#----------------------------------------------------------
def curva(x0, y0, x1, y1):
    xm = (x0 + x1) // 2;
    ym = (y0 + y1) /  2; # float

    # realiza o experimento
    ani = Aniversario(xm,T)
    fxm = ani.mean()
    ani = None # evita 'loitering': essencial quando o espaço é importante

    # base da recursão
    if x1 - x0 < GAP or abs(ym - fxm) < ERR:
        # desenha a linha entre [x0,y0] e [x1, y1]
        plt.plot([x0,x1],[y0,y1], 'b-')
        return None

    # desenhe recursivamente a curva entre [x0,y0] e [xm,fm]
    curva(x0, y0, xm, fxm);

    # desenhe o ponto [xm,fm]
    plt.plot(xm, fxm, 'go') # green bolinha

    # desenhe recursivamente a curva entre [xm,fxm] e [x1,y1]
    curva(xm, fxm, x1, y1);
```

24.11 Colecionador de figurinhas

Coupon collector's problem

Suponha que temos álbum de `n` figurinhas que são numeradas de 0 a `n-1`.

Quantas figurinhas temos que comprar para completar o álbum?

Modelagem

Suporemos que cada figurinha é igualmente provável de ser comprada.

Código

```
import random

class Colecionador:
    #------------------------------------------
    def __init__(self, n, t):
        self.n = n
        self.t = t
        no_figurinhas = 0
        for i in range(t):
            no_figurinhas += self.experimento()
        self.n_medio = no_figurinhas/t

    #------------------------------------------
    def mean(self):
        return self.n_medio

    #-----------------------------------------
    def experimento(self):
        '''(Colecionador) -> int

        Sorteia figurinhas uniformmente ao acaso até o
        que álbum esteja preenchido. O método retorna o
        número de figurinhas sorteadas.
        '''
        n = self.n
        album = set()
        no_compradas = 0
        while len(album) != n:
            fig = random.randrange(n)
            no_compradas += 1
            album |= {fig}
        return no_compradas
```

Cliente

```python
from colecionador import Colecionador

#-------------------------------------------------------
def main(argv=None):
    n = int("digite o número de figurinhas no álbum:")
    t = int("digite o número de teste a serem executados")

    # realize os t testes para preencher um álbum com n figurinhas
    exp = Colecionador(n, t)

    # média aritmética
    print("número de figurinhas compradas =", exp.estimativa())
```

Cálculo do valor esperado

Seja T o número de figurinhas a serem compradas para completarmos o álbum. Seja t_i o número de figurinhas que compramos entre obtermos a $(i-1)$-ésima e a i-ésimo figurinha distintas. T e t_i são variáveis aleatórias e $T = t_1 + t_2 + \cdots + t_n$. A probabilidade p_i de conseguirmos uma figurinha distinta, dado que já conseguimos $i-1$ figurinhas distintas é $(n-(i-1))/n$. Seja $q_i = 1 - p_i = 1/(i-1)$. Assim, o valor esperado de t_i é dado por

$$\begin{aligned} E(t_i) &= 1p_i + 2p_iq_i + 3p_iq_i^2 + \cdots \\ &= p_i(1 + 2q_i + 3q_i^2 + 4q_3^3 + \cdots \\ &= p_i 1/(1-q_i)^2 = 1/p_i. \end{aligned}$$

Desta forma, temos que

$$\begin{aligned} E(T) &= E(t_1) + E(t_2) + \cdots + E(t_n) \\ &= n/n + n/(n-1) + n/(n-2) + \cdots n/1 = n \times H(n) < n\ln(n+1). \end{aligned}$$

Portanto, temos que comprar aproximadamente $n \ln n$ figurinhas para completarmos um álbum de n figurinhas (supondo que a probabilidade ...).

24.12 Problema conforto dos estudantes

Problema

Queremo distrbuir os estudantes na sala de maneira que fiquem confortavelmente sentados, sem um monte de gente ao lado.

Podemos imaginar também que queremos distribuir os alunos na sala de aula para uma prova.

Para isso vamos usar uma função `h()` que atribui aleatoriamente uma carteira a cada estudante que chega na porta.

Suponha que tenhamos `n` estudantes e `m` carteiras. Essa função pode ser até algo do tipo

```
def h(estudante, m):
    ```(str, int) -> int
    Recebe o nome de um ou uma estudante e retorna
    um número de carteira para o/a estudante sentar.
    ```
    random.seed(nome)
    return random.randrange(m)
```

É possível que o/a estudantes recebe o número de uma carteira já sorteada. Nesse caso o/a estudante deverá sondar as carteiras seguintes até encontrar uma vazia.

```
carteira = h(estudante,m)
while sala[carteira] == OCUPADA:
      carteira += 1
      if carteira == m: cateira = 0
sala[carteira] = OCUPADA # recebe o/a estudante
```

Qual o tamanho médio dos grupos que surgirão na sala. Será que todos e todas ficarão distribuidos em grupos pequenos. Por exemplo, para `n = 20` e `m = 40` podemos obter

```
% python main.py 20 40
xxxxxx  xxxxx x       x x  xx x xx     x|

no. grupos: 8
média sondagens em busca bem-sucedida: 1.65
média sondagens em basca  malsucedida: 2.17
```

A intuição diz que todos devem ficar mais ou menos bem distribuídos e não devemos formar grupos grande **na média**. Essa é uma ideia que está na implementação de dicionários e conjuntos em Python (e em muitas outeas linguagens).

A distribuição depende certamente da razão $\alpha = \mathtt{n}/\mathtt{m}$. Esse valor é conhecido pelo nome de **fator de carga** (*load factor*). Dado `n` podemos fixar o valor de `m` para que o número de consultas ou sondagens em alguma busca não seja *na média* nao maior que 5. Esse valor é `O(1)`.

Fato.Supondo que a nossa função `h()` é "bem aleatória', e que α está entre 0 e 1 mas não muito perto de 1, o número médio de sondagens em buscas bem-sucedidas é aproximadamente

$$\frac{1}{2}\Big(1 + \frac{1}{(1-\alpha)}\Big)$$

e o número médio de sondagens em buscas malsucedidas é aproximadamente

$$\frac{1}{2}\Big(1 + \frac{1}{(1-\alpha)^2}\Big)$$

Exemplo: quando $\alpha = 0.5$, temos aproximadamente 1.5 sondagens por busca bem-sucedida e aproximadamente 2.5 sondagens por busca malsucedida.

Exemplo: quando $\alpha = 0.25$, temos aproximadamente 1.16 sondagens por busca bem-sucedida e aproximadamente 1.39 por busca malsucedida.

Não mantendo α "grande" garantimos um desempenho esperado constante para implementações de dicionários e conjuntos.

Programa

O programa `main.py` no diretório `py/dicionario` simula esse fenômeno através de classe `Sala`. Não vale a pena escrever o código na aula. Está uma bagunça e meio errado em alguns cálculos, mas nada muito grave. Mas acho que executar pode trazer alguma intuição.

```
python main.py
Uso: python3 main.py n m
    n = número de alunos
    m = número de carteiras

python main.py 300 800
   x    xx     xxxxxxxxx xxx  x    x     xx  xxxx  x x    x          xxx  x   x  x  xx xxxxxxx
 xxxx  xxx    x  x   x x x   xxxxx    xxxx  x x   x  x    x    x        x   xx         xxxxx
            xxx  xxx xxxxxxxxxxx   x  xxxxx xxxxx xx    x    xx    x    xxxx              x  x x
 x   xx       x  x  x x    x xx   x               xxx   x  x            x    xx   xxx   xx  x
xx         xxxx  xxxxx x  x x   xxxxx xxx         x xx       xxxxx    xx  xxxx     xxx x     xx
   x   x x          xxxxxxx        x    x    x          xx x   x  x  xx xx          x   x     x
x xx x   x      x x x          xx       x     x x  x    x xxxx     x x        x x   xxx    x
xxxxx xx  x  x     x  xx xxxx  xxxxx  x x x |

no. grupos: 146
média sondagens em busca bem-sucedida: 1.34
média sondagens em busca  malsucedida: 1.87
```

Classe

```
import numpy as np
import random

VAZIA   = False
OCUPADA = True

class Sala:
    #-------------------------------------------
    def __init__(self, n, m):
```

```python
    '''(Sala, int, float, int)
    É esperado que n/m esteja entre 0 e 1, mas não muito perto de 1.
    '''
    self.n    = n  # número de estudantes
    self.m    = m  # numero de carteiras
    self.sala = np.full(m, VAZIA) # sala de aula
    self.distribua_estudantes()
    self.estatisticas()

#-----------------------------------------------
def __str__(self):
    s = ''
    for i in range(self.m):
        s += 'x' if self.sala[i] == OCUPADA else ' '
    return s+"|"

#-----------------------------------------------
def no_grupos(self):
    return self.grupos

#-------------------------------------------
def media_malsucedidas(self):
    return self.fracasso

#-------------------------------------------
def media_bemsucedidas(self):
    return self.mean_sucesso

#------------------------------------------
def distribua_estudantes(self):
    n = self.n
    m = self.m
    sala = self.sala
    sucesso = 1
    for i in range(n):
        carteira = random.randrange(m)
        while sala[carteira] != VAZIA:
            carteira += 1
            sucesso  += 1
            if carteira == m: carteira = 0
        sucesso += 1
        sala[carteira] = OCUPADA
    self.mean_sucesso = sucesso/n

#-------------------------------------------
def estatisticas(self):
    sala = self.sala
    m    = self.m
    anterior = VAZIA
    grupos = 0
```

```
fracasso  = 0
for i in range(self.m):
    atual = sala[i]
    if anterior == VAZIA and atual == OCUPADA:
        grupos += 1
        k =  2
    elif anterior == OCUPADA and atual == OCUPADA:
        k += 1
    else:
        k = 1
    fracasso += k
    anterior  = atual

if sala[0] == OCUPADA and sala[-1] == OCUPADA:
    grupos -= 1
self.grupos = grupos
self.fracasso = fracasso/m
```