# Aula 21: 01/NOV/2018

## Aula passada

Programação dinâmica: *longest commom subsequence* (LCS)

## Hoje

Mais programação dinâmica. Comentar como das strings `s` e `t` e da matriz `opt[][]`, construída para encontramos o comprimento de uma`lcs(s,t)`, podemos encontra a string.

Modelos computacionais têm um papel fundamental em pesquisa científica moderna (slides)

Simulação pode ajudar a validar análise matemática, pode validar comportamento de software, pode ser a única opcão: Euler's Method scene in Hidden Figures.

Paradoxo dos aniversários para mostra arcabouço de simulação.

## Longest commom subsequence

*Longest common subsequence* (LCS) = subsequência comum máxima.

### Problema

Dadas duas strings `s` e `t`, encontrar uma subsequência comum máxima se `s` e `t`.

Uma **subsequência** é uma sequência que aparece na mesma ordem relativa, mas não é necessariamente contígua. Por exemplo, `"abc"`, `"abg"`, `"bdf"`, `"aeg"`, `"acefg"`, etc são subsequências de `"abcdefg"`.

## Programação dinâmica

- Dynamic Programming

Método para resolução de problemas complexos transforma o problema original em uma coleção de subproblemas mais simples, resolvendo cada subproblema uma única vez e armazenando os seus resultados. Da próxima vez que o mesmo subproblema é encontrado, a solução pré computada é utilizada, economizando tempo mas com um gasto (esperamos que modesto) de memória.

A subsequência comum máxima pode ser encontrada construindo-se uma tabela onde os valores das maiores subsequências até um determinado tamanho são armazenadas. A resolução dos subproblemas segue uma política *bottom-up*.

```
opt[i][j] = len(lcs(s[0:i],t[0:j]))

opt[i][j] = 0,                              se i == 0 ou j == 0
opt[i][j] = opt[i-1][j-1] + 1,              se s[i-1] == t[j-1]
opt[i][j] = max(opt[i-1][j], opt[i][j-1]), em caso contráriioa
```

```python
# Implementação de programação dinâmica para o problema LCS
# Computa o comprimento do LCS para todo subproblema

def lcs(s, t):
    '''(str, str) -> str

    Recebe uma string s de comprimento m e uma string t de
    comprimento n e retorna a matriz opt com o comprimento dos lcs
    para todo subproblema.
    '''
    # encontre os comprimentos dos strings
    m = len(s)
    n = len(t)

    # define a matriz para armazenar os valores da PD.
    opt = [[0]*(n+1) for i in range(m+1)] # crie_matriz(m+1, n+1, 0)

    # compute as entradas da matriz opt[][] de uma maneira 'bottom up'
    # Note: opt[i][j] contém o comprimento de uma LCS de s[0:i] e t[0:j]
    for i in range(m+1):
        for j in range(n+1):
            if s[i-1] == t[j-1]:
                opt[i][j] = opt[i-1][j-1] + 1
            else:
                opt[i][j] = max(opt[i-1][j] , opt[i][j-1])

    # opt[m][n] contains the length of LCS of s[0:n] & t[0:m]
    return opt

#-------------------------------------------------------------
def get_lcs(s, t, opt):
    '''(str, str, matriz) -> str

    Recebe uma string s e uma string t e a matriz opt com o
    comprimento dos lcs para todo subproblema e retorna uma
    lcs de s e t.
    '''
    lcs = ''
    m, n = len(s), len(t)
    i, j  = m, n
    while i > 0 and j > 0:
        if s[i-1] == t[j-1]:
            lcs = s[i-1] + lcs
            i -= 1
            j -= 1
        elif opt[i-1][j] >= opt[i][j-1]:
            i -= 1
        else:
            j -= 1
    return lcs
```

## Problema (paradoxo do aniversário)

### Motivação

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 com 40 alunos?

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 assumimos que 365 possíveis 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 $n \leq 365$, ele é dado 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 |

### Modelagem

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)

## Código

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

O código utiliza a classe nativa `set`. Podemos usar `list`, mas `set` é mais eficiente.

```
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(365) # valor entre 0 e 364
            if data in aniversarios:
               return True
            aniversarios |= {data} # união de conjuntos, com lista aniversarios.append(data)
        return False
```

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

```
from aniversario import Aniversario

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

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

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


#---------------------------------------
def prob(n):
    '''
    Retorna a probabilidade teorica.
    '''
    prob_c = 1
    for i in range(n):
        prob_c *= (1 - i/DATAS)
    return 1 - prob_c
```

# Apêndice

## Sets

Python also includes a data type for sets. A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the set() function can be used to create sets. Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary, a data structure that we discuss in the next section.

```
>>> # Demonstrate set operations on unique letters from two words
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                              # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # letters in both a and b
{'a', 'c'}
>>> a ^ b                              # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```