Estrutura de Dados é <3

Fernando Masanori @fmasanori

about.me/fmasanori

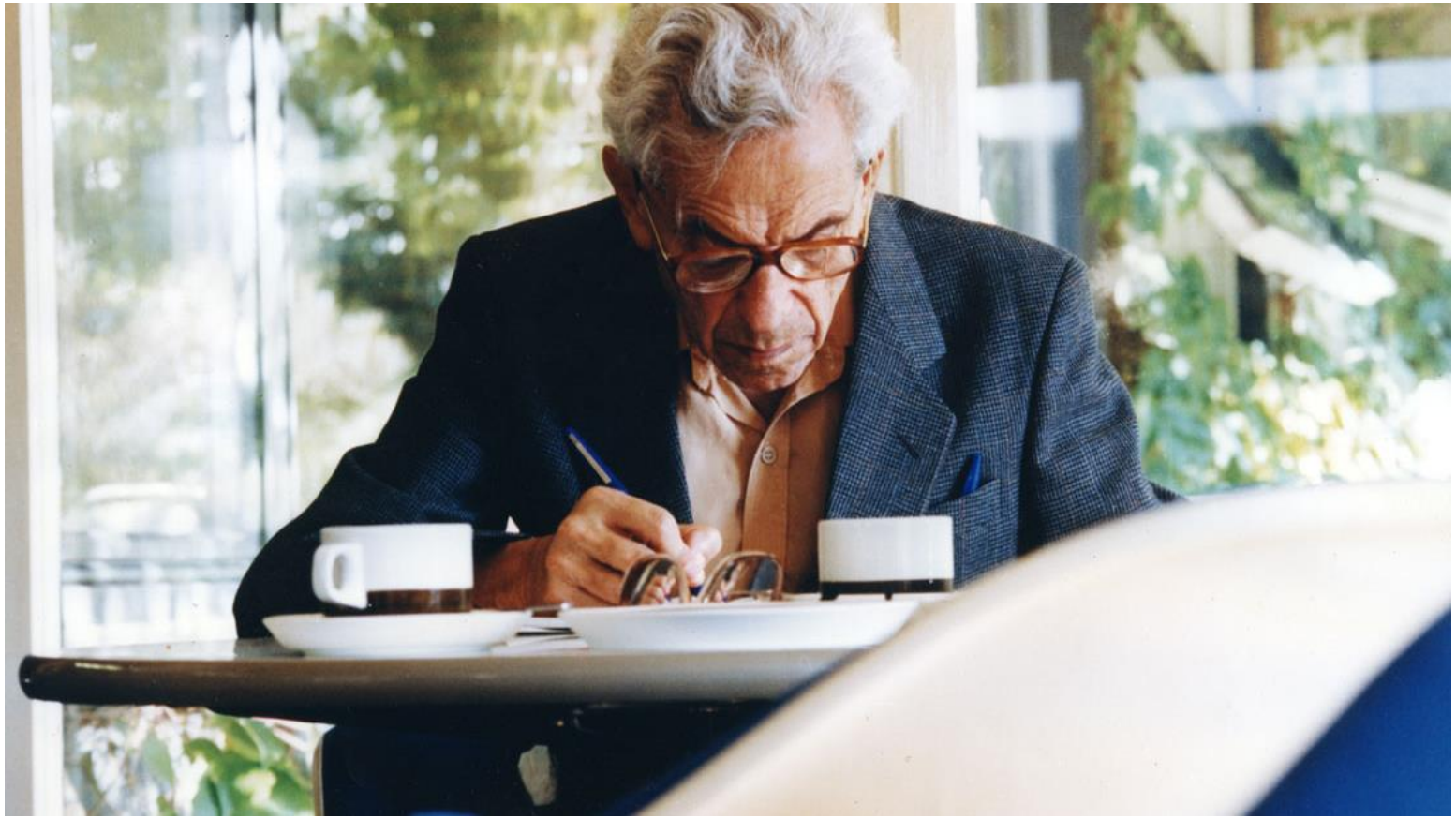
github.com/fmasanori/ED ou bit.ly/PythonED

The most common fault in computer classes is to emphasize the rules of specific programming languages, instead of to emphasize the algorithms that are being expressed in those languages. D. Knuth interview at People of ACM, June, 2014.

"Busca binária é um algoritmo notoriamente difícil de programar corretamente. Somente dezessete anos depois da invenção do algoritmo a primeira versão correta do programa foi publicada!"
[Steven Skiena, The Algorithm Design Manual]

```
def busca_binaria(x, v):
    global cont
    e = -1
    d = len(v)
    while e < d-1:
        m = (e + d) // 2
        cont = cont + 1
        if v[m] < x:
            e = m
        else:
            d = m
    return d
```

"You don't have to believe in God, but you should believe in The Book." [Paul Erdös]



```
def mergesortInterativo(v):
    b = 1
    n = len(v)
    while b < n:
        p = 0
        while p + b < n:
            r = p + 2*b
            if r > n: r = n
            v[p:r] = merge(v[p:p+b], v[p+b: r])
            p = p + 2 * b
        b = 2*b
    return v
```

```
def mergesort(v):
    if len(v) <= 1: return v
    else:
        m = len(v) // 2
        e = mergesort(v[:m])
        d = mergesort(v[m:])
        return merge(e, d)
```

"A Computação se apoia sobre três pernas: a correção, a eficiência e a elegância."
[Imre Simon]

```
def fib(n):
    print (f'fib({n})')
    if n <= 2: return 1
    return fib(n-1) + fib(n-2)

print (fib(10))
#dá para calcular fib(100) nesta palestra?
```

```
dic = {}
def fib(n):
   if n <= 2: return 1
   if n not in dic: dic[n] = fib(n-1) + fib(n-2)
   return dic[n]
print (fib(100))
```

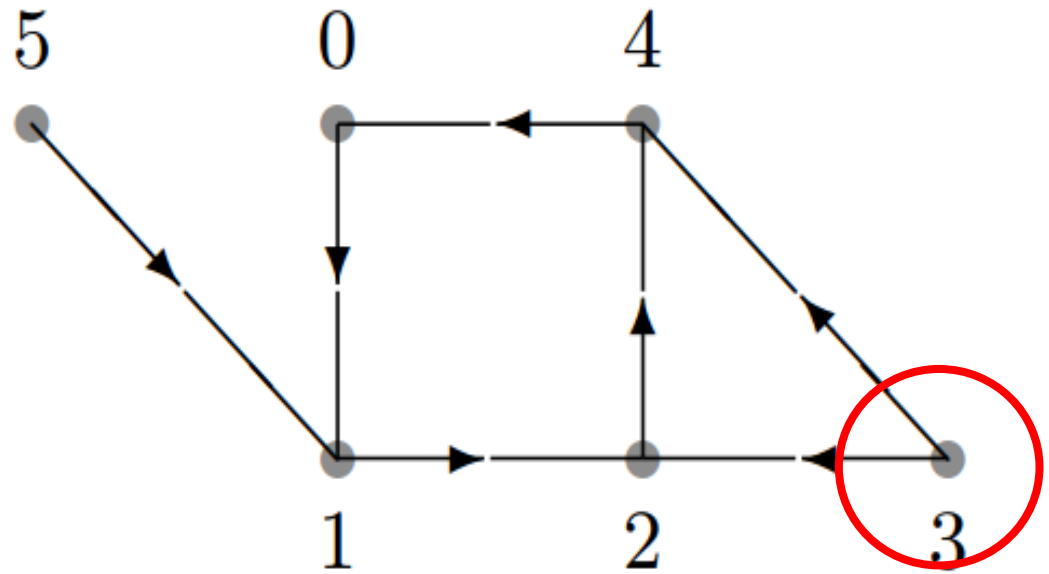
```
from functools import lru_cache


@lru_cache(maxsize=None)
def fib(n):
    print (f'fib({n})')
    if n <= 2: return 1
    return fib(n-1) + fib(n-2)


print (fib(10))
```

```
def BemFormada(s):
    p = []
    for c in s:
        if c == ')':
            if p[-1] == '(': p.pop()
            else: return False
        elif c == '}':
            if p[-1] == '{': p.pop()
            else: return False
        else:
            p.append(c)
    return len(p) == 0
print (BemFormada('(){()})'))
print (BemFormada('({)}'))
```

	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



	0	1	2	3	4	5
d	2	3	1	0	1	6

```
def Distancias(n, origem):
    d = [-1] * n
    d[origem] = 0
    f = []
    f.append(origem)
    while len(f) > 0:
        x = f[0]
        del f[0]
        for y in range(n):
            if A[x][y] == 1 and d[y] == -1:
                d[y] = d[x] + 1
                f.append(y)
    return d
```



"Um bom algoritmo é como uma faca afiada: ele faz o que dele se espera com o mínima de esforço. Usar um algoritmo errado para resolver um problema é como tentar cortar um bife com uma chave de fenda: você pode até mesmo conseguir um resultado aceitável, mas você gastará muito mais esforço que o necessário e é pouco provável que o resultado será esteticamente agradável." [Th. Cormen, Ch. Leiserson, R. Rivest, Introduction to Algorithms]

```
def boyermoore(p, t):
    m = len(p), n = len(t)
    if m > n: return -1
    pulo = [m for k in range(256)]
    for k in range(m - 1): pulo[ord(p[k])] = m-k-1
    pulo = tuple(pulo)
    k = m - 1
    while k < n:
        j = m - 1
        i = k
        while j >= 0 and t[i] == p[j]:
            j -= 1
            i -= 1
        if j == -1: return i + 1
        k += pulo[ord(t[k])] #dou "pulos" > 1
    return -1
```

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult." [C.A.R.Hoare]

```
def quicksort(v):
    if len(v) <= 1: return v

    pivô = v[0]
    iguais  = [x for x in v if x == pivô]
    menores = [x for x in v if x <  pivô]
    maiores = [x for x in v if x >  pivô]
    return quicksort(menores) + iguais + quicksort(maiores)
```

"As famílias felizes parecem-se todas; as famílias infelizes são infelizes cada uma à sua maneira."

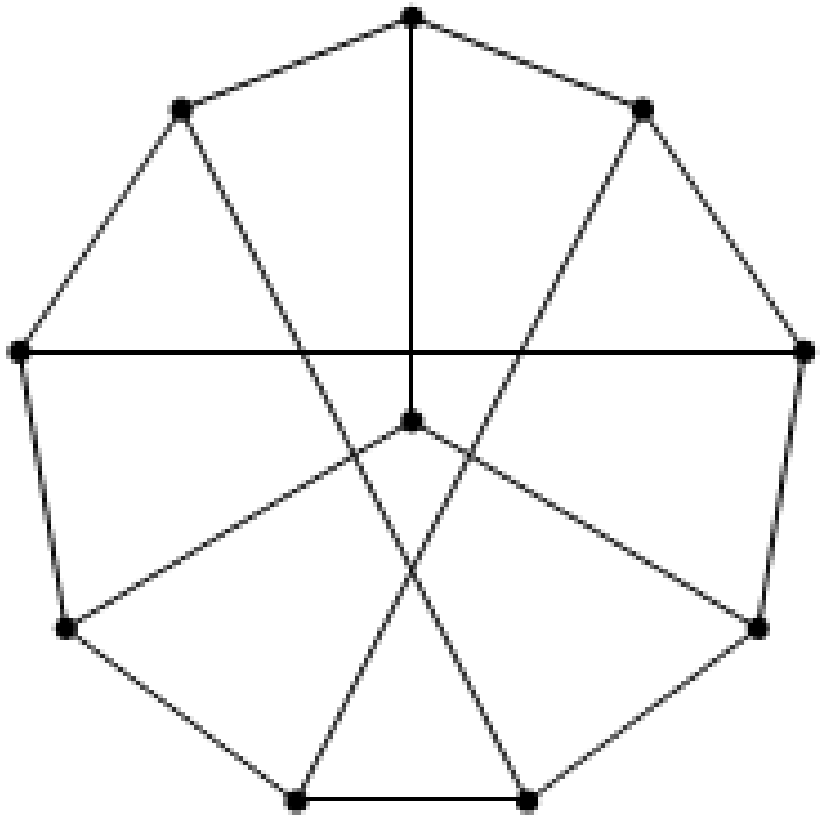
```
def seleção(v):
    r = []
    while v:
        m = min(v)
        r.append(m)
        v.remove(m)
    return r
```

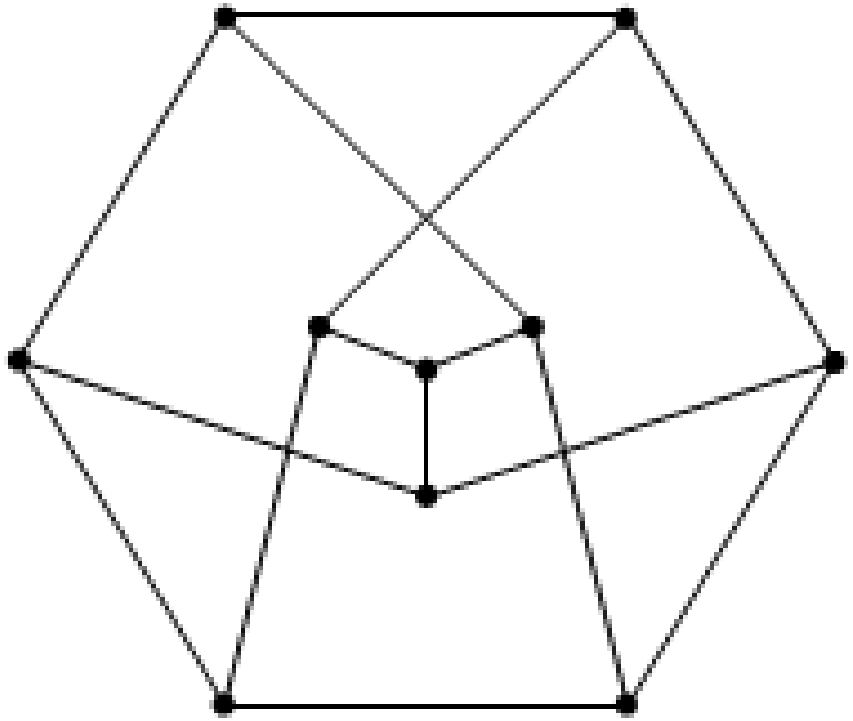
```
def inserção(v):
    for j in range(1, len(v)):
        x = v[j]
        i = j - 1
        while i >= 0 and v[i] > x:
            v[i+1] = v[i]
            i = i - 1
        v[i + 1] = x
    return v
```

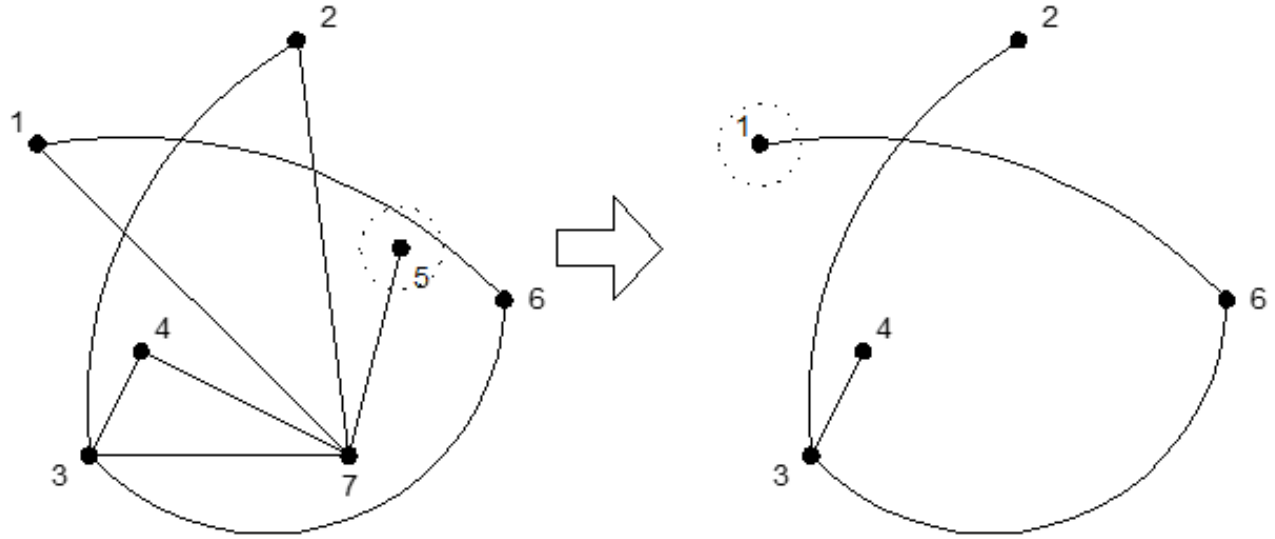
"Muitas vezes não há maneira melhor de resolver um problema que tentar todas as possíveis soluções. Esta abordagem, chamada busca exaustiva, é quase sempre lenta, mas às vezes ela é melhor que nada." [Ian Parberry, Problems on Algorithms]

```
def enumerações(items):
    n = len(items)
    s = [0]*(n+1)
    k = 0
    while True:
        if s[k] < n:
            s[k+1] = s[k] + 1
            k += 1
        else:
            s[k-1] += 1
            k -= 1
        if k == 0:
            break
        else:
            lista = []
            for j in range(1, k+1):
                lista.append(items[s[j]-1])
            yield lista
```

Teoria dos Grafos

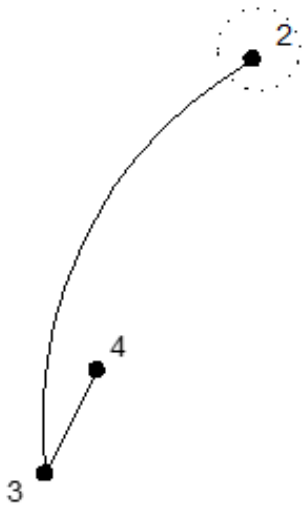






S={5}

S={5, 1}



S={5, 1, 2}

S={5, 1, 2, 4}

Minimum Degree Greedy for MIS

```
def mínimo():
  m = removidos.index(False)
  for v in g:
    if removidos[v]: continue
    if len(g[v]) < len(g[m]): m = v
  return m
def remove_vizinhos(v):
  removidos[v] = True
  for x in g[v]:
    removidos[x] = True
    g[x] = []
    for y in g:
      if x in g[y]:
        g[y].remove(x)
s = []
while False in removidos:
  v = mínimo()
  s.append(v)
  remove_vizinhos(v)
print (s)
```

about.me/fmasanori

github.com/fmasanori/ED

bit.ly/PythonED