Problemes de complexitat

January 17, 2019

1 Problemes de Complexitat

1.1 Algorísmica I

1.1.1 Universitat de Barcelona - Grau d'enginyeria informàtica

Les solucions d'aquest notebook poden no ser òptimes, s'han triat només per criteris pedagògics i de demostració del càlcul de la complexitat. Les solucions no són doncs les dels algorismes, que es donen des de l'inici sinó la solució al càlcul de complexitat. S'anima a intentar fer el càlcul, i només després mirar les solucions.

Ordre invers Tenim una llista ordenada (no necessàriament amb un ordre natural) de mida n. El que volem ara és donar-li la volta, és a dir, tenir l'ordre invers.

```
In [25]: def reverse_order(arr):
             size = len(arr)
             if(size==1):
                 return arr
             elif(size==2):
                 listAux = [arr[1], arr[0]]
                 return listAux
             else:
                 leftArr = arr[:size//2]
                 rightArr = arr[size//2:]
                 revLeftArr = reverse_order(leftArr)
                 revRightArr = reverse_order(rightArr)
                 return revRightArr + revLeftArr
```

```
In [26]: reverse_order([1,2,3,4,5,6,7,8])
```

```
Out[26]: [8, 7, 6, 5, 4, 3, 2, 1]
```

```
In [27]: reverse_order([1,3,4,6,7,12,15,45,65])
```

```
Out[27]: [65, 45, 15, 12, 7, 6, 4, 3, 1]
```

```
In [28]: reverse_order(["cat", "dog", "fish"])
```

```
Out[28]: ['fish', 'dog', 'cat']
```

Sol·lució (dóna dos clics aquí per veure-la)
Suposem algorismes la complexitat dels quals podem escriure com:

1. $T(n) = 3T(n/3) + \sqrt{n}$
2. $T(n) = 3T(n/2) + n^2$
3. $T(n) = 4T(n/2) + n^2$

Calcula la seva complexitat mitjançant el teorema Master
Sol·lució (dóna dos clics aquí per veure-la)
Norma en el infinit Definim la norma a l'infinit d'una matriu A quadrada nxn com:

$$\|A\|_\infty = max_{1\leq j\leq n}\sum_{k=1}^{n}|A_{j,k}|$$

```
In [29]: import math
         def infinity_norm(A):
             inf_norm=0
             for j in range(len(A)):
                 row_sum = 0
                 for k in range(len(A[j])):
                     row_sum += abs(A[j][k])
                 if(inf_norm < row_sum):
                     inf_norm = row_sum
             return inf_norm
```

```
In [30]: infinity_norm([[3,2,1],
                        [1,4,-4],
                        [-2,-4,-1]])
```

```
Out[30]: 9
```

```
In [31]: infinity_norm([[0,0,0],
                         [0,0,0],
                         [0,0,0]])
```

```
Out[31]: 0
```

Sol·lució (dóna dos clics aquí per veure-la)
La jugada de Bernoulli En aquest problema tirarem una moneda n cops. Suposarem que una moneda només pot treure cara (C) o creu (X). Volem saber totes les situacions possibles quan tirem la moneda els n cops. Per exemple, per a $n = 2$ les possibilitats són (C, C), (C, X), (X, C), (X, X)

```
In [32]: def bernoulli_possibilities(n):
             import itertools
             a=lst = list(set(itertools.product(("X","C"),repeat=n)))
             return(a)
```

```
In [33]: bernoulli_possibilities(3)
```

```
Out[33]: [('C', 'X', 'C'),
          ('C', 'C', 'C'),
          ('X', 'X', 'X'),
          ('C', 'C', 'X'),
          ('X', 'C', 'X'),
          ('X', 'C', 'C'),
          ('C', 'X', 'X'),
          ('X', 'X', 'C')]
```

```
In [34]: bernoulli_possibilities(4)
```

```
Out[34]: [('X', 'X', 'X', 'C'),
          ('X', 'C', 'C', 'C'),
          ('C', 'C', 'C', 'C'),
          ('X', 'X', 'X', 'X'),
          ('X', 'C', 'C', 'X'),
          ('C', 'C', 'X', 'X'),
          ('X', 'C', 'X', 'C'),
          ('X', 'X', 'C', 'X'),
          ('X', 'C', 'X', 'X'),
          ('C', 'X', 'C', 'C'),
          ('C', 'C', 'X', 'C'),
          ('C', 'C', 'C', 'X'),
          ('C', 'X', 'C', 'X'),
          ('C', 'X', 'X', 'X'),
          ('X', 'X', 'C', 'C'),
          ('C', 'X', 'X', 'C')]
```

Sol·lució (dóna dos clics aquí per veure-la)
Residus quadràtics Volem trobar els elements r tal que $x^2 = r \ (mod\ m)$

```
In [35]: def quadratic_residue(m):
             q_residues = set()
             for x in range(m):
                 r = (x**2)%m
                 if(r!=0):
                     q_residues.add(r)
             return q_residues
```

```
In [36]: quadratic_residue(15)
```

```
Out[36]: {1, 4, 6, 9, 10}
```

perque 1 mod 15 = 1 i 1 = 1ˆ2; perque 25 mod 15 = 10 i 25 = 5ˆ2; perque 64 mod 15 = 4 i 64 = 8ˆ2 etc

Considerem que treballem amb nombres petits, i per tant les operacions de comparació i aritmètiques tenen un cost de $\mathcal{O}(1)$

Sol·lució (dóna dos clics aquí per veure-la)

Fusió de llistes Donades dues llistes A, B de mida n i m respectivament llistes volem fusionar-les sense alterar la llista inicial. Una manera de fer-ho és la següent:

In [37]:
```
def fusio(A, B):
    novaA=A[:]
    for i in range(len(B)):
        novaA.append(B[i])
    return novaA
```

In [38]:
```
A = [1,3,5,4]
B = [3,2,1,1]
print(fusio(A,B))
print(A)
print(B)
```

```
[1, 3, 5, 4, 3, 2, 1, 1]
[1, 3, 5, 4]
[3, 2, 1, 1]
```

Sol·lució (dóna dos clics aquí per veure-la)

Eliminar duplicats Suposem que tenim una llista amb n elements $[a_1, a_2, ..., a_n]$. Volem generar una nova llista amb el mateix ordre però amb només una còpia de cada element: la primera que surt a la llista. Un algorisme per fer açò és:

In [39]:
```
def delete_duplicates(A):
    newList = []
    for i in range(0,len(A)):
        actual_elem = A[i]
        if(actual_elem != 'repetit'):
            newList.append(actual_elem)
            for j in range(i, len(A)):
                if(A[j] == actual_elem):
                    A[j] = 'repetit'
    return newList
```

In [40]:
```
delete_duplicates([1,1,2,3,4,5,6,7,5,2,9])
```

Out[40]:
```
[1, 2, 3, 4, 5, 6, 7, 9]
```

Sol·lució (dóna dos clics aquí per veure-la)

O(n3)

Producte dels $log(n)$ **primers nombres** Donat un n enter volem saber quin és el producte dels enters des d'1 fins a l'enter més gran menor que $log(n)$

In [41]:
```
from math import log
def multiplication(n):
    i=1
    product = 1
    while(i<=log(n)):
        product *= i
        i+=1
    return product
```

```
In [42]: multiplication(120)
```

```
Out[42]: 24
```

Sol·lució (dóna dos clics aquí per veure-la)

Comprovar si un nombre és primer Sigui n un nombre, volem saber si és primer o no. Per fer-ho només necessitem saber si te algún divisor menor que $\sqrt{n}$. Si no en té no serà primer.

```
In [43]: from math import sqrt
         def check_prime_number(n):
             prime = True
             sqrtn = int(sqrt(n))
             i=2
             while i<=sqrtn and prime:
                 if(n%i==0):
                     prime = False
                 i+=1
             return prime
```

```
In [44]: check_prime_number(9)
```

```
Out[44]: False
```

Sol·lució (dóna dos clics aquí per veure-la)

Llista de repeticions Volem saber quants cops surt cada nombre si fem totes les multiplicacions possibles d'1 fins a n tres cops, és a dir, el següent algorisme:

```
In [45]: def count_repetitions(n):
             dict_of_reps = {}
             for i in range(1,n+1):
                 for j in range(1,n+1):
                     for k in range(1,n+1):
                         if (i*j*k) not in dict_of_reps:
                             dict_of_reps[(i*j*k)] = 1
                         else:
                             dict_of_reps[(i*j*k)] +=1
             return dict_of_reps
```

Retorna només els nombres que surten com a key i el nombre de cops com a value.

```
In [46]: count_repetitions(4)
```

```
Out[46]: {1: 1,
          2: 3,
          3: 3,
          4: 6,
          6: 6,
          8: 7,
          9: 3,
```

```
    12: 9,
    16: 6,
    18: 3,
    24: 6,
    32: 3,
    27: 1,
    36: 3,
    48: 3,
    64: 1}
```

Sol·lució (dóna dos clics aquí per veure-la)