Recursividade: Strassen e Quickselect

Algoritmo de Strassen O algoritmo de Strassen preve uma pequena economia ao realizar a multiplicação matricial. A importância deste algoritmo é que durante anos, a ninguém ocorreu que alguma otimização pudesse ser feita sobre o algoritmo canônico. Strassen percebeu que em $r = a.b$ (a e b matrizes conformáveis) se ambas fossem divididas em 4 (com cortes ao meio), para obter a resposta, ao invés de fazer 8 multiplicações, eram necessárias apenas 7, acompanhe:

- a é dividida em 4: a_{11}, a_{12}, a_{21} e a_{22}
- idem para b
- agora são calculadas 7 sub-matrizes recursivamente, usando diversas parcelas envolvendo a_{ij} e b_{km}.
- finalmente, o resultado é obtido executando operações elementares (adição e subtração) sobre as 7 sub-matrizes.

Um detalhe importante é que a se a matriz a multiplicar não tiver dimensões iguais a uma potência de 2, basta acrescentar zeros nas linhas ou nas colunas para atingir esta condição. Tais linhas e colunas podem ser retiradas depois sem alterar o resultado numérico da operação.

Acompanhe o algoritmo e se quiser efetue a multiplicação matricial canônica para conferir o resultado correto do algoritmo de Strassen.

```
#strassen em Python, usando o pacote numpy
import numpy as np
import random as ra
import time
tam=512
a=np.zeros((tam,tam),float)
for i in range(0,tam):
  for j in range(0,tam):
    a[i][j]=ra.random()
b=np.zeros((tam,tam),float)
for i in range(0,tam):
  for j in range(0,tam):
    b[i][j]=ra.random()

def marra(a,b):
  tam=len(a)
  r=np.zeros((tam,tam),float)
  for i in range(tam):
    for j in range(tam):
      olha=0
      for k in range(tam):
        olha=olha+a[i][k]*b[k][j]
      r[i][j]=olha
  return(r)

def strassen(x,y):
  n=len(x)
  if n>4: # ou n>1
    me=int(n/2)
    a11=x[0:me,0:me]
    a12=x[0:me,me:n]
    a21=x[me:n,0:me]
    a22=x[me:n,me:n]
    b11=y[0:me,0:me]
    b12=y[0:me,me:n]
    b21=y[me:n,0:me]
    b22=y[me:n,me:n]
    m1=strassen((a11+a22),(b11+b22))
    m2=strassen((a21+a22),b11)
    m3=strassen(a11,(b12-b22))
    m4=strassen(a22,(b21-b11))
    m5=strassen((a11+a12),b22)
    m6=strassen((a21-a11),(b11+b12))
    m7=strassen((a12-a22),(b21+b22))
    c=np.zeros((n,n),int)
    c[0:me,0:me]=m1+m4-m5+m7
    c[0:me,me:n]=m3+m5
    c[me:n,0:me]=m2+m4
    c[me:n,me:n]=m1-m2+m3+m6
    return c
  else:
    return marra(x,y) # ou x*y
ini=time.time()
strassen(a,b)
fim=time.time()
print('Demora de strassen',fim-ini)
```

Dados reais: Eis um possível desempenho real destas funções: Para matrizes de 256 reais, o algoritmo de Strassen (com $n > 4$) demorou 13.8 seg e o algoritmo canônico 18.5 seg. Já para matrizes de 512 reais, os tempos foram de 89.6 e 150.3 segundos respectivamente. Fica de fora a análise da função primitiva do numpy-python `dot(a,b)` que tem um desempenho maravilhoso: para os casos citados demorou 0.002 e 0.015 segundos.

Quickselect Trata-se de descobrir a mediana de um conjunto de dados desordenados, sem ser necessário ordená-los. A referência é "Algoritmos para Leigos" de Mueller, John e Massaron, Luca, pág. 334.

Seja calcular a mediana de uma lista (desordenada) de números. Da estatística, sabe-se que a mediana de uma distribuição é o valor que ocupa a posição central DEPOIS que a distribuição é ordenada. (No caso da ordenação ter um número par de elementos, a mediana é a média entre os 2 valores centrais). Esta definição aponta para uma solução simples: ordenar a sequência e depois tomar o elemento central. Só que a ordenação é uma operação custosa (custa $O(n.logn)$ no mínimo, e o que se pretende aqui é achar a mediana sem precisar ordenar nada e pagando apenas $O(n)$. Pode-se batizar este algoritmo como Quickselect, ou o algoritmo do k-ésimo valor. Os passos do algoritmo recursivo são:

1. A chamada é $QS(lista, k)$
2. O algoritmo escolhe um pivot (aleatório) e divide a lista em 2: à esquerda os menores do que o pivot e à direita os maiores do que o pivot.
3. Achar o comprimento das 2 listas. Se o $tam(esquerdo) > k$, então a mediana está na lista esquerda. Aplicar $QS(esquerdo)$.
4. subtrair de k o tamanho da lista esquerdo
5. Calcular o número de duplicações da mediana, fazendo $tam(lista) - (tam(esquerdo) + tam(direito))$
   - Se o número de duplicações é maior do que k, achou-se a mediana
   - senão, remover o número de duplicações de k ($novok = k - duplicacoes$) e aplicar $QS(direito, novok)$.

Em python:

```
from random import randint, random, choice
import numpy as np
import sys
import time
tam=100000
a=np.zeros((tam+1),float)
for i in range(1,tam+1):
  a[i]=random()*100000
def quickselect(series,k):
  pivot=choice(series)
     #retorna um randomico da lista
  left, right = list(),list()
  for item in series:
    if item<pivot:
      left.append(item)
    if item>pivot:
      right.append(item)
  lenght_left=len(left)
  if lenght_left > k:
    return quickselect(left,k)
  k=k-lenght_left
  dupli=len(series)-(lenght_left+len(right))
  if dupli>k:
    return float(pivot)
  k=k-dupli
  return quickselect(right,k)
ini=time.time()
print(quickselect(a,(tam//2)+1))
fim=time.time()
print("tempo quickselect ",fim-ini)
```

```
def partition(arr, begin, end):
  pivot = begin
  for i in range(begin+1, end+1):
    if arr[i] <= arr[begin]:
      pivot = pivot + 1
      arr[i],arr[pivot]=arr[pivot],arr[i]
  arr[pivot],arr[begin]=arr[begin],arr[pivot]
  return pivot

def quicksort(array, begin=0, end=None):
  if end is None:
    end = len(array) - 1
  def _quicksort(array, begin, end):
    if begin >= end:
      return
    pivot = partition(array, begin, end)
    _quicksort(array, begin, pivot-1)
    _quicksort(array, pivot+1, end)
  return _quicksort(array, begin, end)

ini=time.time()
print(quicksort(a))
print(a[1+(tam//2)])
fim=time.time()
print("tempo canonico ",fim-ini)
```

Eis a chamada completa para obter a mediana:

```
def mediana(se):
  if len(se)%2 != 0:
    return quickselect(se,len(se,len(se)//2)
  else:
    le=quickselect(se,(len(se)-1)//2)
    ri=quickselect(se,(len(se)+1)//2)
    return (le+ri)/2
```

☞ Para você fazer

Planeje uma execução comparativa entre dois ambientes: No primeiro o objetivo é achar a mediana de um vetor grande (digamos 100.001 reais) primeiro ordenando e depois listando o 50.000^o número. Depois, faça a mesma medida de tempo usando esta função acima. Conclua sobre o resultado obtido.

Para seu controle em uma situação real, num computador meio lerdinho, em um vetor de 10.000 reais, o quickselect demorou 0.016 seg e o algoritmo canônico, 0.144 seg. Para um vetor de 50.000 reais os tempos foram 0.076 e 0.808 segundos.