Big Data e Big Problem

Normalmente si può trovare come definizione di big data la seguente:

```
I dati sono troppo grandi per entrare in memoria/disco rigido
```

Esiste però un altro problema Big:

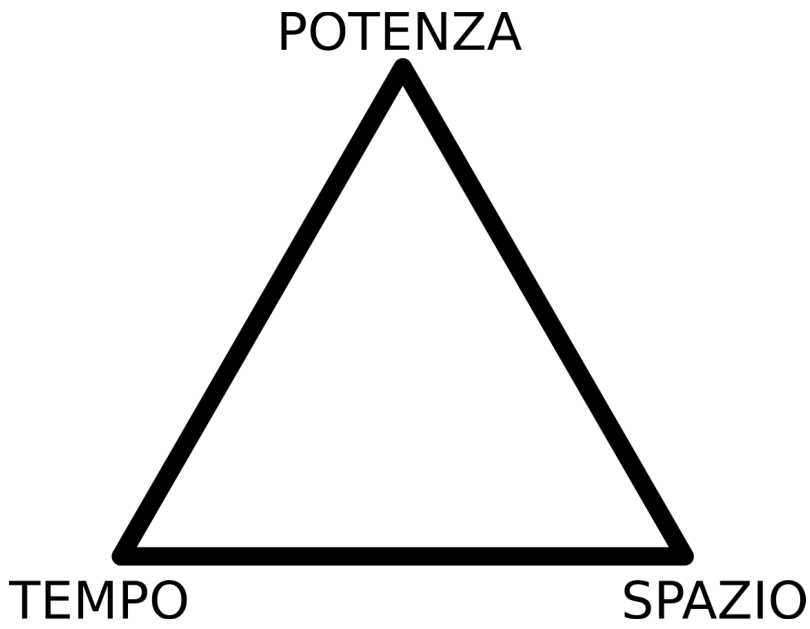
```
Anche su dati piccoli, il mio modello potrebbe richiedere più
risorse di quelle a mia disposizione
```

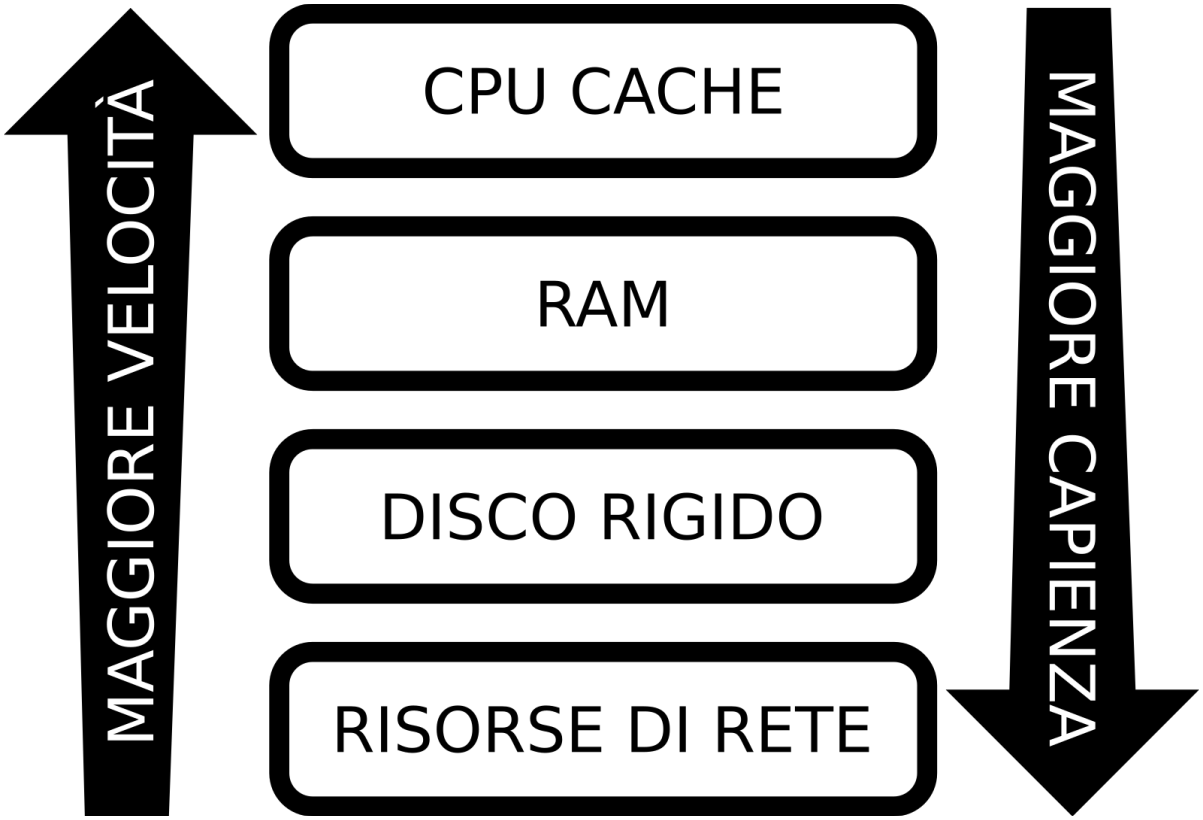
Nei big data dobbiamo fare delle scelte

Semplificando, ci sono 3 variabili nel nostro sistema:

- il tempo che siamo disponibili ad aspettare
- lo spazio su RAM che siamo disponibili ad allocare
- l'ammontare di informazioni che vogliamo ottenere (statistical power)

Tanto più ottimizziamo uno di questi parametri, tanto più ci rimettiamo negli altri due.


POTENZA
TEMPO
SPAZIO


MAGGIORE VELOCITÀ
CPU CACHE
RAM
DISCO RIGIDO
RISORSE DI RETE
MAGGIORE CAPIENZA

Inoltre, vedremo poi, che tanto più ottimizzeremo il nostro codice, tanto più difficile sarà mantenerlo nel tempo. La comprensibilità del nostro codice ha un costo quantificabile!

Questo schema non tiene conto anche di altri fattori:

- il tradoff fra memoria su disco e su RAM
- il tempo che serve all'analista per produrre il codice

un problema di esempio - rinominare i file di una collezione audio

Ovviamente si tratta di un problema giocattolo.

Cosa succederebbe se i file non sono qualche centinaio, ma qualche milione?

E se per ciascun file non dovessi solo rinominare, ma fare analisi complicate?

I principi rimangono gli stessi, soltanto più estremi.

In [1]:

```
! rm -fR notebookfiles/fakeaudio

! mkdir -p notebookfiles/fakeaudio

! mkdir -p notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall

! touch notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall/The\ Thin\ Ice.MP3

! touch notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall/Another\ Brick\ In\ Th
e\ Wall\ -\ Part\ 1.MP3

! mkdir -p notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth/

! touch notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth/My\ Innermost\ Ap
ocalypse.MP3

! touch notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth/NONTOCCARE.TXT

! echo "un testo inutile" > notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebir
th/NONTOCCARE.TXT
```

In [2]:
```
! ls -lR notebookfiles/fakeaudio/
```

```
notebookfiles/fakeaudio/:
totale 8
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 Albums
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 OST

notebookfiles/fakeaudio/Albums:
totale 4
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 PinkFloyd

notebookfiles/fakeaudio/Albums/PinkFloyd:
totale 4
drwxrwxr-x 2 claudia claudia 4096 gen 27 17:18 TheWall

notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall:
totale 0
-rw-rw-r-- 1 claudia claudia 0 gen 27 17:18 'Another Brick In The Wall - Part
1.MP3'
-rw-rw-r-- 1 claudia claudia 0 gen 27 17:18 'The Thin Ice.MP3'

notebookfiles/fakeaudio/OST:
totale 4
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 TheBindingOfIsaac

notebookfiles/fakeaudio/OST/TheBindingOfIsaac:
totale 4
drwxrwxr-x 2 claudia claudia 4096 gen 27 17:18 Rebirth

notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth:
totale 4
-rw-rw-r-- 1 claudia claudia  0 gen 27 17:18 'My Innermost Apocalypse.MP3'
-rw-rw-r-- 1 claudia claudia 17 gen 27 17:18  NONTOCCARE.TXT
```

In [3]:
```
import os  # pip install os
```

In [4]:
```
directory = "./notebookfiles/fakeaudio/"
for basepath, listdir, listfiles in os.walk(directory):
    print()
    print(basepath)
    print(listdir)
    print(listfiles)
```

```
./notebookfiles/fakeaudio/
['OST', 'Albums']
[]

./notebookfiles/fakeaudio/OST
['TheBindingOfIsaac']
[]

./notebookfiles/fakeaudio/OST/TheBindingOfIsaac
['Rebirth']
[]

./notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth
[]
['My Innermost Apocalypse.MP3', 'NONTOCCARE.TXT']

./notebookfiles/fakeaudio/Albums
['PinkFloyd']
[]

./notebookfiles/fakeaudio/Albums/PinkFloyd
['TheWall']
[]

./notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall
[]
['Another Brick In The Wall - Part 1.MP3', 'The Thin Ice.MP3']
```

In [5]:

```
directory = "./notebookfiles/fakeaudio/"

for basepath, listdir, listfiles in os.walk(directory):

    for filename in listfiles:

        new_filename = filename.lower()

        os.rename( os.path.join(basepath, filename),

                   os.path.join(basepath, new_filename) )
```

In [6]:
```
!ls -lR notebookfiles/fakeaudio/  # elenca i files di tutte le subdirectories
```

```
notebookfiles/fakeaudio/:
totale 8
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 Albums
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 OST

notebookfiles/fakeaudio/Albums:
totale 4
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 PinkFloyd

notebookfiles/fakeaudio/Albums/PinkFloyd:
totale 4
drwxrwxr-x 2 claudia claudia 4096 gen 27 17:18 TheWall

notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall:
totale 0
-rw-rw-r-- 1 claudia claudia 0 gen 27 17:18 'another brick in the wall - part
1.mp3'
-rw-rw-r-- 1 claudia claudia 0 gen 27 17:18 'the thin ice.mp3'

notebookfiles/fakeaudio/OST:
totale 4
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 TheBindingOfIsaac

notebookfiles/fakeaudio/OST/TheBindingOfIsaac:
totale 4
drwxrwxr-x 2 claudia claudia 4096 gen 27 17:18 Rebirth

notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth:
totale 4
-rw-rw-r-- 1 claudia claudia  0 gen 27 17:18 'my innermost apocalypse.mp3'
-rw-rw-r-- 1 claudia claudia 17 gen 27 17:18  nontoccare.txt
```

qui però sto anche cambiando il nome a tutti i file che sono presenti nelle directory, anche se non sono dei file audio!

In [7]:

```
! rm -fR notebookfiles/fakeaudio

! mkdir -p notebookfiles/fakeaudio

! mkdir -p notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall

! touch notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall/The\ Thin\ Ice.MP3

! touch notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall/Another\ Brick\ In\ Th
e\ Wall\ -\ Part\ 1.MP3

! mkdir -p notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth/

! touch notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth/My\ Innermost\ Ap
ocalypse.MP3

! touch notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth/NONTOCCARE.TXT

! echo "un testo inutile" > notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebir
th/NONTOCCARE.TXT
```

In [8]:

```
directory = "./notebookfiles/fakeaudio/"

for basepath, listdir, listfiles in os.walk(directory):

    for filename in listfiles:

        if filename.lower().endswith('mp3'):

            new_filename = filename.lower()

            os.rename(os.path.join(basepath, filename),

                      os.path.join(basepath, new_filename))

            print(os.path.join(basepath, filename))
```

```
./notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth/My Innermost Apocalyps
e.MP3
./notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall/Another Brick In The Wall -
Part 1.MP3
./notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall/The Thin Ice.MP3
```

In [9]:
```
! ls -lR notebookfiles/fakeaudio/
```

```
notebookfiles/fakeaudio/:
totale 8
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 Albums
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 OST

notebookfiles/fakeaudio/Albums:
totale 4
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 PinkFloyd

notebookfiles/fakeaudio/Albums/PinkFloyd:
totale 4
drwxrwxr-x 2 claudia claudia 4096 gen 27 17:18 TheWall

notebookfiles/fakeaudio/Albums/PinkFloyd/TheWall:
totale 0
-rw-rw-r-- 1 claudia claudia 0 gen 27 17:18 'another brick in the wall - part
1.mp3'
-rw-rw-r-- 1 claudia claudia 0 gen 27 17:18 'the thin ice.mp3'

notebookfiles/fakeaudio/OST:
totale 4
drwxrwxr-x 3 claudia claudia 4096 gen 27 17:18 TheBindingOfIsaac

notebookfiles/fakeaudio/OST/TheBindingOfIsaac:
totale 4
drwxrwxr-x 2 claudia claudia 4096 gen 27 17:18 Rebirth

notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth:
totale 4
-rw-rw-r-- 1 claudia claudia  0 gen 27 17:18 'my innermost apocalypse.mp3'
-rw-rw-r-- 1 claudia claudia 17 gen 27 17:18  NONTOCCARE.TXT
```

Queste sono tre operazioni fondalmentali per l'analisi dati:

- iterazione lazy
- map (ripeti un'operazione su tutti gli elementi)
- filter (seleziona solo una parte degli elementi)

altri tipi di operazioni che discuteremo saranno:

- reduce (comporre insieme gli elementi)
- functional programming

Iterazione Lazy

che cosa intendiamo con iterazione **lazy**?

le operazioni non vengono compiute finchè il risultato non è richiesto!

In Python questa cosa è gestita da degli oggetti chiamati **iteratori**.

Sono gli oggetti su cui faccio i cicli **for**.

Un iteratore può essere percorso una volta sola!

Questo è controintuitivo: se provo a fare un ciclo for su di una lista, lo posso fare quante volte voglio

In [10]:

```
lista = [1, 2, 3]
print("--- prima iterazione ---")
for elemento in lista:
    print(elemento)
print("--- seconda iterazione ---")
for elemento in lista:
    print(elemento)
```

```
--- prima iterazione ---
1
2
3
--- seconda iterazione ---
1
2
3
```

ma se provo a farlo su di un file, lo posso leggere una volta sola!

Se volessi rileggerlo, dovrei aprirlo di nuovo!

In [11]:
```
directory = "./notebookfiles/fakeaudio/OST/TheBindingOfIsaac/Rebirth/"
filename = "NONTOCCARE.TXT"
position = os.path.join(directory, filename)
with open(position) as file:
    print("--- prima iterazione ---")
    for line in file:
        print(repr(line))
    print("--- seconda iterazione ---")
    for line in file:
        print(repr(line))
```

```
--- prima iterazione ---
'un testo inutile\n'
--- seconda iterazione ---
```

Python ce lo nasconde, ma in realtà ogni volta che iteriamo sulla lista lui crea un nuovo iteratore che scorre la lista e poi scompare.

Possiamo farlo esplicitamente con il comando **iter**

In [12]:
```
lista = [1, 2, 3]
iteratore_lista = iter(lista)
print("prima iterazione")
for elemento in iteratore_lista:
    print(elemento)
print("seconda iterazione")
for elemento in iteratore_lista:
    print(elemento)
```

```
prima iterazione
1
2
3
seconda iterazione
```

Sottolineare quando le iterazioni non necessitano di caricare l'intero dataset è importante perchè non è sempre vero.

Supponiamo di voler calcolare tutte le combinazioni di elementi di una sequenza: non possiamo risolvere questo problema senza tenere in memoria l'intera sequenza!

Map

Un tipo di operazione molto frequente sulle sequenze è il cosiddetto **mapping**, ovvero applicare una funzione a tutti gli elementi di una lista, uno alla volta ed indipendentemente dagli altri.

Ad esempio, avendo una serie di numeri, potrei voler prendere il quadrato di ciascuno.

In [13]:
```
numeri = [0, 1, 2, 3, 4, 5, 6]
quadrati = []
for numero in numeri:
    quadrato = numero **2
    quadrati.append(quadrato)

print(quadrati)
```

```
[0, 1, 4, 9, 16, 25, 36]
```

Questo può essere espresso in modo più conciso con una *comprehension*, che è funzionalmente identica al ciclo visto prima, ma più sintetica

In [14]:
```
numeri = [0, 1, 2, 3, 4, 5, 6]
quadrati = [x**2 for x in numeri]
print(quadrati)
```

```
[0, 1, 4, 9, 16, 25, 36]
```

il concetto di **map** è un'astrazione di questo procedimento.

Python fornisce una funzione, chiamata appunto **map**, che prende una funzione ed un iteratore e ritorna un iteratore i cui elementi sono il risultato dell'operazione

In [15]:
```
def quadrato(n):
    return n**2

numeri = [0, 1, 2, 3, 4, 5, 6]
quadrati = map(quadrato, numeri)
print(quadrati)
```

```
<map object at 0x7fadf4397438>
```

ricordiamoci che il risultato delle operazioni sugli iteratori, quando possibile, è a sua volta un iteratore!

siamo noi che dobbiamo esplicitamente **concretizzare** l'iterazione

In [16]:
```
list(quadrati)
```

Out[16]:
```
[0, 1, 4, 9, 16, 25, 36]
```

ricordiamoci che l'iterazione è compiuta una volta sola, quindi se vogliamo il risultato dobbiamo salvarcelo alla prima concretizzazione!

In [17]:
```
list(quadrati)
```

Out[17]:
```
[]
```

Filter

in modo simile il concetto di **filter** è piuttosto semplice: seleziono un sottoinsieme dei miei dati, generando un secondo iteratore.

In [18]:
```
numeri = [-2, -1, 0, 1, 2]
positivi = []
for numero in numeri:
    if numero>0:
        positivi.append(numero)

print(positivi)
```

```
[1, 2]
```

in modo simile all'operazione di **map**, anche l'operazione di **filter** ha un costrutto nel linguaggio tramite le *comprehension*

In [19]:
```
numeri = [-2, -1, 0, 1, 2]
positivi = [x for x in numeri if x > 0]
print(positivi)
```

```
[1, 2]
```

ed esattamente come prima, abbiamo una funzione che prende una funzione di filtro (che ci dice se l'elemento è accettabile o no) e la applica ad un operatore

In [20]:
```
def is_positive(n):

    return n > 0

positivi = filter(is_positive, numeri)

print(list(positivi))
```

```
[1, 2]
```

Reduce

questa operazione combina gli elementi di un iteratore in un elemento unico

In [21]:
```
numeri = [1, 2, 3, 4]

totale = 0

for numero in numeri:

    totale += numero

print(totale)
```

```
10
```

come per casi precedenti, esiste una funzione preesistente per effettuare le operazioni di riduzione

In [22]:
```
from functools import reduce

def somma(a, b):

    return a + b

numeri = [1, 2, 3, 4]

totale = reduce(somma, numeri, 0)

print(totale)
```

```
10
```

Questo tipo di operazioni è così comune che ci sono una serie di operazioni predefinite:

- **sum** per la somma
- **min** e **max** per il minimo e massimo

e così via

una tipica riduzione, che useremo molto, è la stima delle frequenze.

In [23]:
```
from collections import Counter

numeri = [ 1, 1, 1, 2, 2, 3, 3, 4, 4, 4, 4, 4 ]

Counter( numeri )
```

Out[23]:
```
Counter({1: 3, 2: 2, 3: 2, 4: 5})
```

Una proprietà importante delle riduzioni è che i risultati si possono combinare: dati i conteggi su due serie, posso sommare insieme i due conteggi ed ottenere i conteggi totali fra le due serie

Map Reduce

il famoso metodo MAP-REDUCE è una combinazione di queste idee:

- prendo una sequenza, la divido in sottosequenze
- invio le sequenze a diversi computer
- compio una riduzione su ciascuna sottosequenza
- raccolgo le sottosequenze e le combino insieme
- tutto questo fatto in modo ricorsivo

In [ ]: