Table of Contents

Fondamenti di Programmazione

Andrea Sterbini

lezione 9 - 3 novembre 2022

In [1]:
```
1 %load_ext nb_mypy
```

```
Version 1.0.4
```

RECAP:

- annotazioni di tipo e type checkers:
  - **mypy**: statico (in Spyder e Jupyter)
  - **typeguard**: run-time
  - **pyre-check** (Facebook)
  - **pytype** (Google)
  - **pyright** (Microsoft)
- dizionari ed istogramma

Anagrammi

Due parole sono anagrammi se **contengono le stesse lettere lo stesso numero di volte**

- Soluzione 1: contiamo le lettere

In [6]:
```python
def isAnagramma(testo : str, testo1 : str) -> bool:
    # conto i caratteri nelle due stringhe    O(n)
    # confronto i due dizionari               O(n)
    def conta_lettere(stringa : str) -> dict[str,int]:
        frequenze : dict[str,int] = {}
        for c in stringa:
            if c in frequenze:
                frequenze[c] += 1
            else:
                frequenze[c] =1
        return frequenze
    frequenze1 = conta_lettere(testo)
    frequenze2 = conta_lettere(testo1)
    return frequenze1 == frequenze2

isAnagramma('onepaper', 'paperino')
```

Out[6]:
```
False
```

Forma canonica

Altre definizioni, sono anagrammi se:

- **una si ottiene dall'altra per spostamento di lettere**
- oppure **hanno la stessa forma "standardizzata"**
  (o "forma canonica")

NOTA: la forma "canonica" deve essere **unica**

Esempio di riduzione a forma canonica:
formula booleana ==> forma SOP ==> tabella di verità

Come **"standardizzare"** le parole in modo che siano uguali?

- ordiniamo le lettere

In [9]:
```python
def isAnagramma2(testo : str, testo1 : str) -> bool:
    # ordino le due stringhe      O(n log(n))
    # le confronto                O(n)
    ordinata1 = sorted(testo)
    ordinata2 = sorted(testo1)
    return ordinata1 == ordinata2

isAnagramma2('paperone','eaepnrpo')
```

Out[9]: True

Database come liste di dizionari

ciascun dizionario rappresenta una riga

In [10]:
```python
Scheda = dict[str, str]  # una Scheda è un dizionario 'informazione' -> 'valore'
Agenda = list[Scheda]    # una Agenda è una lista di Schede

agenda : Agenda = [
    {'nome': 'Paperino','cognome':'Paolino',      'telefono':'555-1313',  'indirizzo'
    {'nome': 'Gastone', 'cognome':'Paperone',     'telefono':'555-1717',  'indirizzo'
    {'nome': 'Paperon', 'cognome':"de' Paperoni", 'telefono':'555-99999', 'indirizzo'
    {'nome': 'Archimede','cognome':'Pitagorico',  'telefono':'555-11235', 'indirizzo'
    {'nome': 'Pietro',  'cognome':'Gambadilegno', 'telefono':'555-66666', 'indirizzo'
    {'nome': 'Trudy',   'cognome':'Gambadilegno', 'telefono':'555-66666', 'indirizzo'
    {'nome': 'Topolino','cognome':'Mouse',        'telefono':'555-12345', 'indirizzo'
    {'nome': 'Minnie',  'cognome':'Mouse',        'telefono':'555-54321', 'indirizzo'
    {'nome': 'Pippo',   'cognome':"de' Pippis",   'telefono':'555-33333', 'indirizzo'
    ]
```

Ricerca di un record

In [13]:
```python
from typing import Optional
# per cercare un numero di telefono (un record)
# sapendo quale colonna usare e quale valore cerchiamo
def cerca_lineare(ag : Agenda,
                  colonna : str,
                  valore : str) -> Optional[Scheda] :
    # per tutti i record dell'agenda
    for record in ag:
        # se *la colonna esiste ed inoltre contiene il valore cercato*
        if corrisponde_alla_query(record, colonna, valore):
            # torniamo il record
            return record
    # altrimenti alla fine torniamo None
    return None

cerca_lineare(agenda, 'cognome', 'Pitagorico')
```

Out[13]:
```
{'nome': 'Archimede',
 'cognome': 'Pitagorico',
 'telefono': '555-11235',
 'indirizzo': 'colle degli Inventori 1',
 'città': 'Paperopoli'}
```

In [14]:
```python
# per vedere se un record corrisponde alla query (nome della colonna e valore cercato)
def corrisponde_alla_query(riga : Scheda,
                           colonna : str,
                           valore : str) -> bool:
    # la colonna deve esistere
    # il valore deve corrispondere
    return (colonna in riga
            and riga[colonna] == valore)
```

ricerca multicolonna

In [17]:
```python
# una query è un dizionario colonna -> valore
# per cercare su più colonne
def cerca_multicolonna_lineare(
    ag : Agenda, query : Scheda) -> list[Scheda] :
    # IN : agenda, query: coppie colonna/valore (con un dizionario)
    # OUT: lista dei record che soddisfano tutte le condizioni
    # all'inizio l'elenco di record risultante è []
    trovati = []
    # scandiamo l'agenda e per tutti i record
    for record in ag:
        # se *corrispondono alla query*
        if corrisponde_alla_query_multicolonna(
            record, query):
            # aggiungo il record all'output
            trovati.append(record)
    # torno l'elenco di record
    return trovati
```

In [22]:
```python
# per determinare se un record corrisponde ad una query
def corrisponde_alla_query_multicolonna(
    record : Scheda, query : Scheda ) -> bool :
    # per ciascuna delle coppie colonna : valore cercato
    for colonna, cercato in query.items():
        # se colonna NON è nel record e o non ha quel valore
        if (colonna not in record
            or record[colonna] != cercato):
            # torno False
            return False
    # altrimenti alla fine torno True
    return True

Q = { 'città' : 'Topolinia',
      'cognome' : 'Paolino' }
cerca_multicolonna_lineare(agenda,Q)
```

Out[22]:
```
[]
```

Stile di programmazione funzionale

- **all** : torna True se TUTTI i valori sono True
- **any** : torna True se ALMENO un valore è True
- **filter** : torna solo gli elementi per cui è vero un **predicato**
- **map** : torna una "lista" di elementi trasformati

- ...

In [26]:
```python
# per determinare se un record corrisponde ad una query
# versione funzionale
def corrisponde_alla_query_FUN(
    record : Scheda, query : Scheda ) -> bool :
    # è vero che per ogni coppia k,v in query
        # k è in record ed inoltre il valore corrisponde
    return all(
        (chiave in record
         and record[chiave] == valore)
        for chiave, valore in query.items()
    )

R = {'1':'a', '2':'b', '3':'c'}
Q = {'2':'b', '1':'a'}
corrisponde_alla_query_FUN(R, Q)
```

Out[26]: True

In [27]:
```python
# per determinare se un record corrisponde ad una query
# versione con insiemi
def corrisponde_alla_query_SET(
    record : Scheda, query : Scheda ) -> bool :
    # L'intersezione di record e query è == query
    # ovvero query - record == set vuoto
    set_query = set(query.items())
    set_record = set(record.items())
    return set_query - set_record == set()

corrisponde_alla_query_SET(R,Q)
```

Out[27]: True

In [30]:
```python
# per cercare su più colonne con list comprehension
def cerca_multicolonna_lineare_LC(
    ag : Agenda, query : Scheda) -> list[Scheda] :
    return [
        record for record in ag
        if corrisponde_alla_query_SET(
            record, query)
    ]

Q = { 'città' : 'Topolinia',
     'cognome' : 'Gambadilegno'  }
cerca_multicolonna_lineare_LC(agenda,Q)
```

Out[30]:
```
[{'nome': 'Pietro',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'},
 {'nome': 'Trudy',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'}]
```

In [53]:
```python
%nb_mypy Off
def cerca_multicolonna_lineare_filter(
    ag : Agenda, query : Scheda) -> list[Scheda] :
    # filtro i record con la funzione c_a_q
    def c_a_q(record):
        return corrisponde_alla_query_SET(
            record, query)
    return list(filter(c_a_q, ag))

cerca_multicolonna_lineare_filter(agenda,Q)
```

Out[53]:
```
[{'nome': 'Pietro',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'},
 {'nome': 'Trudy',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'}]
```

In [17]:

```python
def cerca_multicolonna_lineare_lambda(ag : Agenda, query : Scheda) -> list[Scheda] :
    # lo stesso si può fare con una lambda
    pass
```

In [60]:
```python
# per ordinare una agenda rispetto ad una colonna
def ordina_rispetto_a_colonna(ag : Agenda, colonna : str) -> Agenda :
    # usiamo sorted con il parametro key
    # che deve tornare il valore rispetto al quale vogliamo ordinare
    def criterio(riga : Scheda) -> str:
        return riga.get(colonna, '')
    return sorted(ag, key=criterio)

ordina_rispetto_a_colonna(agenda, 'città')
```

Out[60]:
```
[{'nome': 'Paperino',
  'cognome': 'Paolino',
  'telefono': '555-1313',
  'indirizzo': 'via dei Peri 113',
  'città': 'Paperopoli'},
 {'nome': 'Gastone',
  'cognome': 'Paperone',
  'telefono': '555-1717',
  'indirizzo': 'via dei Baobab 42',
  'città': 'Paperopoli'},
 {'nome': 'Paperon',
  'cognome': "de' Paperoni",
  'telefono': '555-99999',
  'indirizzo': 'colle Papero 1',
  'città': 'Paperopoli'},
 {'nome': 'Archimede',
  'cognome': 'Pitagorico',
  'telefono': '555-11235',
  'indirizzo': 'colle degli Inventori 1',
  'città': 'Paperopoli'},
 {'nome': 'Pietro',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'},
 {'nome': 'Trudy',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'},
 {'nome': 'Topolino',
  'cognome': 'Mouse',
  'telefono': '555-12345',
  'indirizzo': 'via degli Investigatori 1',
  'città': 'Topolinia'},
 {'nome': 'Minnie',
```

```
 'cognome': 'Mouse',
 'telefono': '555-54321',
 'indirizzo': 'via di M.me Curie 1',
 'città': 'Topolinia'},
{'nome': 'Pippo',
 'cognome': "de' Pippis",
 'telefono': '555-33333',
 'indirizzo': 'via dei Pioppi 1',
 'città': 'Topolinia'}]
```

In [62]:
```python
# per ordinare una agenda rispetto ad una colonna
def ordina_rispetto_a_colonna_L(ag : Agenda, colonna : str) -> Agenda :
    # usiamo sorted con il parametro key
    # che deve tornare il valore rispetto al quale vogliamo ordinare
    return sorted(ag,
                  key=lambda riga: riga.get(colonna, ''))

ordina_rispetto_a_colonna_L(agenda, 'cognome')
```

Out[62]:
```
[{'nome': 'Pietro',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'},
 {'nome': 'Trudy',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'},
 {'nome': 'Topolino',
  'cognome': 'Mouse',
  'telefono': '555-12345',
  'indirizzo': 'via degli Investigatori 1',
  'città': 'Topolinia'},
 {'nome': 'Minnie',
  'cognome': 'Mouse',
  'telefono': '555-54321',
  'indirizzo': 'via di M.me Curie 1',
  'città': 'Topolinia'},
 {'nome': 'Paperino',
  'cognome': 'Paolino',
  'telefono': '555-1313',
  'indirizzo': 'via dei Peri 113',
  'città': 'Paperopoli'},
 {'nome': 'Gastone',
  'cognome': 'Paperone',
  'telefono': '555-1717',
  'indirizzo': 'via dei Baobab 42',
  'città': 'Paperopoli'},
 {'nome': 'Archimede',
  'cognome': 'Pitagorico',
  'telefono': '555-11235',
  'indirizzo': 'colle degli Inventori 1',
  'città': 'Paperopoli'},
 {'nome': 'Paperon',
  'cognome': "de' Paperoni",
```

```
 'telefono': '555-99999',
 'indirizzo': 'colle Papero 1',
 'città': 'Paperopoli'},
{'nome': 'Pippo',
 'cognome': "de' Pippis",
 'telefono': '555-33333',
 'indirizzo': 'via dei Pioppi 1',
 'città': 'Topolinia'}]
```

MA: perchè non costruire un indice?

- usando un dizionario
- con velocità di ricerca molto alta

In [64]:

```python
# un indice è un dizionario valore -> lista di posizioni nella Agenda
Indice = dict[str,list[int]]
# per costruire un indice
def crea_indice(ag: Agenda, colonna : str ) -> Indice:
    # IN agenda, colonna
    # OUT lista che contiene coppie [valori]:
    #     posizioni originali dei record che contengono quel valore
    # all'inizio il dizionario valori:posizioni è vuoto
    indice = {}
    # per ogni record dell'agenda e sua posizione,
    for i,record in enumerate(ag):
        # estraggo dal record il valore per quella colonna
        valore = record.get(colonna, '')
        # se il valore è nell'indice
        if valore in indice:
            # aggiungo la posizione all'elenco delle posizioni di quel valore
            indice[valore].append(i)
        # altrimenti
        else:
            # aggiungo il valore : [posizione]
            indice[valore] = [i]
    # ritorno il dizionario valori: posizioni
    return indice

indice_cognome = crea_indice(agenda, 'cognome')
indice_cognome
```

Out[64]:

```
{'Paolino': [0],
 'Paperone': [1],
 "de' Paperoni": [2],
 'Pitagorico': [3],
 'Gambadilegno': [4, 5],
 'Mouse': [6, 7],
 "de' Pippis": [8]}
```

In [65]:
```
## Visto che ci siamo costruiamo tutti gli indici
colonne = ['nome','cognome','telefono','indirizzo','città']
indici : dict[str , Indice] = {
    colonna : crea_indice(agenda, colonna)
        for colonna in colonne
}

indici = {}
for colonna in colonne:
    indici[colonna] = crea_indice(agenda, colonna)


indici
```

Out[65]:
```
{'nome': {'Paperino': [0],
  'Gastone': [1],
  'Paperon': [2],
  'Archimede': [3],
  'Pietro': [4],
  'Trudy': [5],
  'Topolino': [6],
  'Minnie': [7],
  'Pippo': [8]},
 'cognome': {'Paolino': [0],
  'Paperone': [1],
  "de' Paperoni": [2],
  'Pitagorico': [3],
  'Gambadilegno': [4, 5],
  'Mouse': [6, 7],
  "de' Pippis": [8]},
 'telefono': {'555-1313': [0],
  '555-1717': [1],
  '555-99999': [2],
```

Ricerca tramite indici

- ricerca singola
- ricerca multipla

In [67]:

```python
## Per cercare con ricerca singola avendo l'indice
def cerca_con_indice(
    ag : Agenda, index : Indice, valore : str):
    # se il valore è nell'indice
    # torno tutti i record nelle posizioni indicate
    if valore not in index:
        return []
    else:
        return [ ag[i] for i in index[valore] ]

cerca_con_indice(agenda, indici['cognome'], 'Gambadilegno')
```

Out[67]:

```
[{'nome': 'Pietro',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'},
 {'nome': 'Trudy',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'}]
```

In [70]:

```python
Indici = dict[str, Indice]
def cerca_con_indici(
    ag : Agenda, indexes : Indici, query: Scheda ) -> Agenda :
    # mi servono tutte le Schede
    # che appaiono negli indici dove sono i valori cercati
    posizioni = None
    # per tutte le coppie colonna : valore nella query
    for colonna,valore in query.items():
        # se il nome colonna non appare negli indici
        if colonna not in indexes:
            # torno []    (oppure aggiungo l'indice?)
            return []
        # se il valore non appare nell'indice della colonna
        if valore not in indexes[colonna]:
            # torno []
            return []
        # altrimenti prendo l'insieme delle righe e lo interseco
        righe_ok = indexes[colonna][valore]
        if posizioni is None:
            posizioni = set(righe_ok)
        else:
            posizioni &= set(righe_ok)
    # alla fine torno la lista di righe rimaste nell'insieme
    return [ ag[i] for i in posizioni ]

cerca_con_indici(agenda, indici,
                 {'cognome':'Gambadilegno', 'nome':'Trudy'})
```

Out[70]:

```
[{'nome': 'Trudy',
  'cognome': 'Gambadilegno',
  'telefono': '555-66666',
  'indirizzo': 'via dei Ladri 13',
  'città': 'Topolinia'}]
```