Algoritmo di rappresentazione dei polinomi simmetrici

1 Algoritmo di rappresentazione dei polinomi simmetrici negli e_i

Lo scopo di questo notebook è implementare l'algoritmo impiegato nella dimostrazione del *Teorema fondamentale dei polinomi simmetrici*, che, dato un campo F, asserisce il seguente isomorfismo:

$$\mathrm{Sym}[X_n] \cong F[e_1, \dots, e_n],$$

dove X_n è l'insieme $\{x_1, \dots, x_n\}$.

Innanzitutto, si sceglie un $n \in \mathbb{N}^+$, che rappresenta il numero di variabili di cui è composto il polinomio simmetrico, e si crea l'anello di polinomi $\mathbb{Q}[x_1, \dots, x_n]$, nel quale vale il *degree lexicographic order* (**deglex**). Vengono infine create delle variabili simboliche a rappresentare i polinomi simmetrici elementari.

[1]:
```
%display latex
n = 5

P = PolynomialRing(QQ, ",".join(f"x_{i}" for i in range(1, n+1)),␣
↪order='deglex')
P.inject_variables()

var(",".join(f"e_{i}" for i in range(n+1)))
```

```
Defining x_1, x_2, x_3, x_4, x_5
```

[1]: $(e_0, e_1, e_2, e_3, e_4, e_5)$

Si definisce il valore che deve assumere la funzione $e(d)$, che rappresenta il polinomio simmetrico e_d, nel seguente modo:

$$e(d) = \begin{cases} 1 & \text{se } d = 0, \\ \displaystyle\sum_{1 \le i_1 < i_2 < \dots < i_d \le n} \underbrace{x_{i_1} \cdots x_{i_d}}_{d \text{ volte}} & \text{altrimenti.} \end{cases}$$

[2]:
```
from functools import reduce
from itertools import combinations
from operator import mul
```

```
def e(d: Integer):
    if d == 0:
        return 1

    return sum(reduce(mul, [P(f"x_{j}") for j in i], 1) for i in␣
↪combinations(range(1, n+1), d))
```

Si definisce una funzione $\beta(\text{exp\_lt})$ che prende in ingresso una tupla ordinata $\text{exp\_lt} \in \mathbb{N}^n$ e restituisce una tupla dello stesso tipo definita nel seguente modo:

$$\beta(\text{exp\_lt})_i = \begin{cases} \text{exp\_lt}_i - \text{exp\_lt}_{i+1} & \text{se } 1 \leq i < n, \\ \text{exp\_lt}_i & \text{altrimenti.} \end{cases}$$

[3]:
```
def beta(exp_lt):
    return [e - exp_lt[i+1] if i != n-1 else e for i, e in enumerate(exp_lt)]
```

Si definiscono due funzioni analoghe, dette e_prod e e_prod_value. La seconda restituisce lo stesso polinomio di e_prod sostituendo ai simboli e_i i corrispettivi valori $e(i)$. Pertanto si definisce solo il valore della prima funzione.

Tale funzione e_prod prende in ingresso una tupla $b \in \mathbb{N}^n$ e restituisce $e^b = {e_1}^{b_1} {e_2}^{b_2} \cdots {e_n}^{b_n}$.

[4]:
```
def e_prod(b):
    return reduce(mul, [eval(f"e_{i+1}")^k for i, k in enumerate(b)], 1)


def e_prod_value(b):
    return reduce(mul, [e(i+1)^k for i, k in enumerate(b)], 1)
```

Si definisce infine la funzione $\text{combination}(\text{poly}(x))$, che prende in ingresso un polinomio simmetrico $\text{poly}(x)$ e ne restituisce la rappresentazione in $F[e_1, \dots, e_n]$, secondo il seguente algoritmo.

- se $\text{poly}(x)$ è 0 o ha grado nullo, la sua rappresentazione in $F[e_1, \dots, e_n]$ è già $\text{poly}(x)$, e quindi la funzione restituisce il polinomio in ingresso senza modificarlo,
- altrimenti, si considera, secondo il *deglex*, il *leading term* di $\text{poly}(x)$, detto lt:
  - detta α la tupla ordinata degli esponenti di lt, si calcola $\beta(\alpha)$, detto b,
  - detto c il coefficiente razionale di lt, si restituisce la rappresentazione simbolica $c \cdot \text{e\_prod(b)}$, a cui si aggiunge ricorsivamente la rappresentazione del polinomio $\text{poly(x)} - c \cdot \text{e\_prod\_value(b)}$, ottenuta reiterando l'algoritmo su di esso.

[5]:
```
def combination(poly):
    if isinstance(poly, Integer) or isinstance(poly, Rational) or poly == 0 or␣
↪poly.degree() == 0:
        return poly

    lt = sorted(list(poly), reverse=True)[0]

    try:
```

```
        b = beta(lt[1].exponents()[0])
        return lt[0] * e_prod(b) + combination(poly - lt[0] * e_prod_value(b))
    except (AttributeError, TypeError):
        raise TypeError("The given polynomial is not of symmetric kind")
```

Si sceglie inoltre un polinomio $\mathrm{poly}(x)$ che *deve* essere simmetrico – qualora non fosse tale, l'algoritmo non terminerà con successo (se terminasse con successo, il polinomio sarebbe combinazione di polinomi simmetrici, e sarebbe dunque anch'esso simmetrico,).

[6]:
```
# Il polinomio deve essere simmetrico...
poly = x_1 + x_2 + x_3 + x_4 + x_5 + (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 -␣
 ↪x_1*x_2*x_3*x_4*x_5)^2
poly
```

[6]:

$$x_1^2x_2^2x_3^2x_4^2x_5^2 - 2x_1^3x_2x_3x_4x_5 - 2x_1x_2^3x_3x_4x_5 - 2x_1x_2x_3^3x_4x_5 - 2x_1x_2x_3x_4^3x_5 - 2x_1x_2x_3x_4x_5^3 + x_1^4 + 2x_1^2x_2^2 + 2x_1^2x_3^2 + 2x_1^2x_4^2 + 2x_1^2x_5^2 + x_2^4 + 2x_2^2x_3^2 + 2x_2^2x_4^2 + 2x_2^2x_5^2 + x_3^4 + 2x_3^2x_4^2 + 2x_3^2x_5^2 + x_4^4 + 2x_4^2x_5^2 + x_5^4 + x_1 + x_2 + x_3 + x_4 + x_5$$

Si calcola adesso la rappresentazione di $\mathrm{poly}(x)$ in funzione dei polinomi simmetrici elementari secondo l'implementazione di combination:

[7]:
```
combination(poly)
```

[7]:

$$e_1^4 - 4\,e_1^2e_2 - 2\,e_1^2e_5 + 4\,e_2^2 + 4\,e_2e_5 + e_5^2 + e_1$$

Data una somma di potenze simmetrica, è possibile implementare un algoritmo di rappresentazione negli e_i secondo le identità di Newton-Girard. Si definisce allora la funzione $\mathrm{newton\_girard}(k)$, che prende in ingresso un $k \in \mathbb{N}$ e restituisce la rappresentazione di $\sum_{i=1}^{n} {x_i}^k$ nel seguente modo:

$$\mathrm{newton\_girard}(k) = \begin{cases} n & \text{se } k = 0, \\ (-1)^{k+1} \cdot \left(ke_k + \sum_{i=1}^{k-1} (-1)^i \,\mathrm{newton\_girard}(i)\, e_{k-1}\right) & \text{altrimenti,} \end{cases}$$

dove si pone $e_i = 0$ per ogni $i > n$.

La funzione è implementata con l'ausilio di un sistema di *caching*, affinché non vengano ricalcolati i valori già calcolati.

[8]:
```
from functools import lru_cache

@lru_cache(maxsize=None)
def newton_girard(k):
    if k == 0:
        return n

    return ((-1)**(k+1) * ((k * eval(f"e_{k}") if k <= n else 0) +
                            sum((-1)**i * newton_girard(i) * eval(f"e_{k - i}")␣
 ↪for i in range(max(1, k-n), k)))).expand()
```

Si definisce infine la funzione sum_of_powers(k), che, dato in ingresso un $k \in \mathbb{N}$, restituisce la rappresentazione di $\sum_{i=1}^{n}(x_i)^k$ secondo la funzione combination già definita.

[9]:
```
def sum_of_powers(k):
    return combination(sum(eval(f"x_{i}")^k for i in range(1, n+1)))
```

Si sceglie infine un $k \in \mathbb{N}$ al quale elevare tutti le variabili della somma.

[10]:
```
k = 8
```

Si computa la rappresentazione di $\sum_{i=1}^{n}(x_i)^k$ prima secondo newton_girard, e poi secondo sum_of_powers, e si verifica che siano uguali.

[11]:
```
a = newton_girard(k)
a
```

[11]:

$$e_1^8 - 8\,e_1^6e_2 + 20\,e_1^4e_2^2 + 8\,e_1^5e_3 - 16\,e_1^2e_2^3 - 32\,e_1^3e_2e_3 - 8\,e_1^4e_4 + 2\,e_2^4 + 24\,e_1e_2^2e_3 + 12\,e_1^2e_3^2 + 24\,e_1^2e_2e_4 + 8\,e_1^3e_5 - 8\,e_2e_3^2 - 8\,e_2^2e_4 - 16\,e_1e_3e_4 - 16\,e_1e_2e_5 + 4\,e_4^2 + 8\,e_3e_5$$

[12]:
```
b = sum_of_powers(k)
b
```

[12]:

$$e_1^8 - 8\,e_1^6e_2 + 20\,e_1^4e_2^2 + 8\,e_1^5e_3 - 16\,e_1^2e_2^3 - 32\,e_1^3e_2e_3 - 8\,e_1^4e_4 + 2\,e_2^4 + 24\,e_1e_2^2e_3 + 12\,e_1^2e_3^2 + 24\,e_1^2e_2e_4 + 8\,e_1^3e_5 - 8\,e_2e_3^2 - 8\,e_2^2e_4 - 16\,e_1e_3e_4 - 16\,e_1e_2e_5 + 4\,e_4^2 + 8\,e_3e_5$$

[13]:
```
a - b
```

[13]:

0

Sia ora:

$$f(x) = x^n + a_{n-1}x^{n-1} \dots + a_0,$$

con radici x_1, x_2, ..., x_n. Allora si definisce **discriminante** di f la seguente produttoria:

$$\Delta(f) = \prod_{1 \le i < j \le n} (x_i - x_j)^2.$$

In particolare, vale la seguente proprietà:

$$\exists\, i \neq j \mid x_i = x_j \iff \Delta(f) = 0.$$

Si verifica facilmente che $\Delta(f)$ è un polinomio simmetrico. Poiché ogni permutazione è una composizione di trasposizioni, è sufficiente verificare che invertendo due radici x_i e x_j il discriminante rimanga invariato:

- i fattori che contengono solo uno tra x_i e x_j mantengono la somma della base invariata o al più cambiano di segno, e dunque, elevando al quadrato, rimangono invariati,

- il fattore che contiene sia x_i che x_j cambia la propria base di segno, ma rimane invariato elevando al quadrato.

Poiché $\Delta(f)$ è un polinomio simmetrico nelle variabili x_1, ..., x_n, per il *Teorema fondamentale dei polinomi simmetrici*, si scrive in modo unico in funzione dei polinomi simmetrici elementari $e_i(x_1, \dots, x_n)$. Tuttavia tali polinomi simmetrici elementari altro non sono che i coefficienti di $f(x)$, a meno del segno. In particolare vale che:

$$e_i(x_1, \dots, x_n) = (-1)^i a_{n-i}, \quad \text{per } 0 \leq i \leq n.$$

Si definiscono quindi i simboli a_0, ..., a_n e si costruisce una funzione $\Delta(n)$ che restituisce il discriminante di un generico polinomio di n-esimo grado dato in ingresso in funzione degli a_i mediante combination.

[14]:
```
var(",".join(f"a_{i}" for i in range(n+1)))

def delta(n):

    f = reduce(mul, (eval(f"(x_{i}-x_{j})")^2 for i, j in combinations(range(1,␣
↪n+1), 2)), 1)
    c = combination(f)

    for i in range(1, n+1):
        if i % 2:
            c = eval(f"c.substitute(e_{i}=-a_{n-i})")
        else:
            c = eval(f"c.substitute(e_{i}=a_{n-i})")

    return c
```

Dunque, per l'n scelto in partenza, il discriminante del generico polinomio di n-esimo grado è il seguente:

[15]:
```
delta(n)
```

[15]:

$$a_1^2a_2^2a_3^2a_4^2 - 4\,a_0a_2^3a_3^2a_4^2 - 4\,a_1^3a_3^3a_4^2 + 18\,a_0a_1a_2a_3^3a_4^2 - 27\,a_0^2a_3^4a_4^2 - 4\,a_1^2a_2^3a_4^3 + 16\,a_0a_2^4a_4^3 + 18\,a_1^3a_2a_3a_4^3 - 80\,a_0a_1a_2^2a_3a_4^3 - 6\,a_0a_1^2a_3^2a_4^3 + 144\,a_0^2a_2a_3^2a_4^3 - 27\,a_1^4a_4^4 + 144\,a_0a_1^2a_2a_4^4 - 128\,a_0^2a_2^2a_4^4 - 192\,a_0^2a_1a_3a_4^4 + 256\,a_0^3a_4^5 - 4\,a_1^2a_2^2a_3^3 + 16\,a_0a_2^3a_3^3 + 16\,a_1^3a_3^4 - 72\,a_0a_1a_2a_3^4 + 108\,a_0^2a_3^5 + 18\,a_1^2a_2^3a_3a_4 - 72\,a_0a_2^4a_3a_4 - 80\,a_1^3a_2a_3^2a_4 + 356\,a_0a_1a_2^2a_3^2a_4 + 24\,a_0a_1^2a_3^3a_4 - 630\,a_0^2a_2a_3^3a_4 - 6\,a_1^3a_2^2a_4^2 + 24\,a_0a_1a_2^3a_4^2 + 144\,a_1^4a_3a_4^2 - 746\,a_0a_1^2a_2a_3a_4^2 + 560\,a_0^2a_2^2a_3a_4^2 + 1020\,a_0^2a_1a_3^2a_4^2 - 36\,a_0a_1^3a_4^3 + 160\,a_0^2a_1a_2a_4^3 - 1600\,a_0^3a_3a_4^3 - 27\,a_1^2a_2^4 + 108\,a_0a_2^5 + 144\,a_1^3a_2^2a_3 - 630\,a_0a_1a_2^3a_3 - 128\,a_1^4a_3^2 + 560\,a_0a_1^2a_2a_3^2 + 825\,a_0^2a_2^2a_3^2 - 900\,a_0^2a_1a_3^3 - 192\,a_1^4a_2a_4 + 1020\,a_0a_1^2a_2^2a_4 - 900\,a_0^2a_2^3a_4 + 160\,a_0a_1^3a_3a_4 - 2050\,a_0^2a_1a_2a_3a_4 + 2250\,a_0^3a_3^2a_4 - 50\,a_0^2a_1^2a_4^2 + 2000\,a_0^3a_2a_4^2 + 256\,a_1^5 - 1600\,a_0a_1^3a_2 + 2250\,a_0^2a_1a_2^2 + 2000\,a_0^2a_1^2a_3 - 3750\,a_0^3a_2a_3 - 2500\,a_0^3a_1a_4 + 3125\,a_0^4$$

Infine si costruisce la funzione evaluate_delta che, dato in ingresso un polinomio $f(x)$ di n-esimo grado, restituisce il valore di $\Delta(f)$, sostituendo agli a_i generici di delta(n) i valori dei coefficienti di f.

[16]:
```
def evaluate_delta(f):
    d = delta(n)

    for i, a in enumerate(f.list()):
        d = eval(f"d.substitute(a_{i}=a)")

    return d
```

Si verifica adesso che per un polinomio con radici multiple il valore di evaluate_delta è precisamente zero:

[17]:
```
f = (x-2)^2*(x-3)*(x-4)*(x-5)
f
```

[17]: $(x-2)^2(x-3)(x-4)(x-5)$

[18]:
```
f.expand()
```

[18]: $x^5 - 16\,x^4 + 99\,x^3 - 296\,x^2 + 428\,x - 240$

[19]:
```
evaluate_delta(f)
```

[19]: 0

Infine, si fa lo stesso con un polinomio con radici distinte, per verificare che evaluate_delta è strettamente diverso da zero.

[20]:
```
g = (x+2)*(x+I)*(x-I)*(x-1)*(x-2)
g
```

[20]: $(x+2)(x+i)(x-i)(x-1)(x-2)$

[21]:
```
g.expand()
```

[21]: $x^5 - x^4 - 3\,x^3 + 3\,x^2 - 4\,x + 4$

[22]:
```
evaluate_delta(g)
```

[22]: -1440000
