4 (2p) (done)

8 November, 2023 15:14

L6.4. 2 punkty Wykaż, że dla dowolnych $k, l \in \mathbb{N}$ oraz $x \in \mathbb{R}$ zachodzi

a)

$$T_{kl}(x) = T_k(T_l(x)).$$

b) Wykorzystaj podaną zależność do opracowania **szybkiego algorytmu** wyznaczania wartości wielomianu Czebyszewa **wysokiego** stopnia niebędącego liczbą pierwszą.

$\cos(n \arccos(\cos y)) = \cos(ny)$

a) Z poprzedniego zadania wiemy że $T_n(x) = \cos(n \arccos x)$.

Zapiszmy x w postaci $x = \cos(y)$. Wówczas łatwo możemy przekształcić do postaci $T_n(\cos(y)) = \cos(ny)$.

Teraz prawą stronę równania które chcemy udowodnić możemy zapisać jako:

$T_k(T_l(\cos y))$

i za pomocą paru prostych przekształceń:

$T_k(T_l(\cos(y))) = T_k(\cos(ly)) = \cos(kly) = T_{kl}(\cos(y)) = T_{kl}(x) = L$ Udowodnione.

* i to nie tylko dla $x \in \langle -1; 1 \rangle$. Dla $x \in (-1; 1)$ wiemy że wielomiany będą miały takie same pochodne ⇒ takie same szeregi Taylora ⇒ równe wszędzie

ew. dowód podobny do 5b (jednoznaczność Lagrange'a)

tj. $P(x) - Q(x)$ ma nieskończoną ilość zer ⇒ $P(x) = Q(x)$ dla wszystkich x

b) Wykorzystaj podaną zależność do opracowania **szybkiego algorytmu** wyznaczania wartości wielomianu Czebyszewa **wysokiego** stopnia niebędącego liczbą pierwszą.

Pomysł podobny do szybkiego potęgowania. Będziemy dużo korzystali z T_2, więc je też zapiszę

Bazowe przypadki:

$$\boxed{T_k = 2x T_{k-1}(x) - T_{k-2}(x)}$$

$n=0$: 1

$n=1$: x

$n=2$: $2x^2 - 1$

i teraz dla dowolnego n:

n parzyste, $n=2k$: $T_{2k}(x) = T_2(T_k(x))$

n nieparzyste, $n=2k+1$: $T_{2k+1}(x) = 2x \cdot T_2(T_k(x)) - T_{2k-1}(x)$; widzimy tu nieefektywność

da się uprościć do bazowych, liczb złożonych i liczb pierwszych (liczonych normalnie)

Więc korzystamy z faktu że nie jest liczbą pierwszą (i będziemy to robić tak długo jak to możliwe - w.p.p. nasz algorytm działa jak zwykły algorytm

```python
# Zwykły Czebyszew dla testów
def Czebyszew(n, x):
    if n == 0:
        return 1
    if n == 1:
        return x
    return 2 * x * Czebyszew(n - 1, x) - Czebyszew(n - 2, x)

# Pomocnicze funkcje
def czy_pierwsza(n):
    return n > 1 and all(n % i != 0 for i in range(2, int(n**0.5) + 1))

def pierwszy_dzielnik(n):
    return next((i for i in range(2, int(n**0.5) + 1) if n % i == 0), n)


# Algorytm wyznaczający n-tą potęgę Czebyszewa dla x ∈ R w czasie O(log n).
# Wykorzystujemy zależność T_kl(x) = T_k(T_l(x))
def FastCzebyszew(n, x):
    # Najpierw bazowe przypadki
    if n == 0:
        return 1
    if n == 1:
        return x
    if n == 2:
        return 2 * x * x - 1
    if n % 2 == 0:
        return FastCzebyszew(2, FastCzebyszew(n // 2, x))
    # Teraz n jest nieparzyste
    # Sprawdzamy czy n jest liczbą pierwszą - jeśli nie to musimy korzystać z normalnego algorytmu
    if czy_pierwsza(n):
        return 2 * x * Czebyszew(n - 1, x) - Czebyszew(n - 2, x)
    # Teraz n jest nieparzyste i złożone
    dzielnik1 = pierwszy_dzielnik(n)
    dzielnik2 = n / dzielnik1
    return FastCzebyszew(dzielnik1, FastCzebyszew(dzielnik2, x))
```

test dla 100 losowo wygenerowanych par

```
Pierwszy ciąg liczb:
[16, 26, 18, 18, 28, 15, 21, 27, 8, 15, 20, 25, 10, 22, 20, 14, 24, 4, 18, 8, 21, 21, 18, 20, 4, 25, 27, 1, 28, 1, 20, 14, 16, 20, 1, 10, 22, 16, 4, 22, 27, 14, 20, 14
, 20, 22, 12, 10, 24, 22, 15, 9, 18, 1, 10, 25, 1, 20, 16, 4, 27, 22, 16, 27, 28, 28, 9, 4, 1, 14, 18, 8, 21, 30, 16, 25, 18, 18, 28, 16, 30, 9, 28, 6, 25, 30, 24, 4,
12, 10, 25, 20, 14, 6, 26, 10, 9, 14, 1, 15]

Drugi ciąg liczb:
[22, 10, 8, 6, 27, 18, 4, 28, 16, 1, 15, 9, 10, 22, 12, 28, 28, 10, 12, 28, 20, 26, 18, 10, 27, 8, 4, 1, 20, 4, 16, 20, 10, 20, 20, 20, 1, 21, 12, 22, 27, 16, 26, 22,
6, 27, 22, 21, 21, 16, 6, 1, 28, 12, 10, 6, 12, 18, 26, 14, 20, 12, 15, 6, 12, 18, 8, 12, 22, 28, 26, 20, 20, 30, 20, 26, 21, 25, 14, 22, 16, 10, 9, 25, 4, 27, 10, 18,
 15, 18, 22, 10, 6, 10, 27, 20, 24, 15, 14, 16]
Brak błędów.
```

i jeszcze porównanie czasu.

```
Czebyszew(30, 30) czas wykonania: 0.4213752746582031 sekundy
FastCzebyszew(30, 30) czas wykonania: 0.0 sekundy
```