Programowanie
przed 3 wykładem

Andrzej Giniewicz

15.03.2024

W tym tygodniu zajmiemy się rekurencjami. Zarazem rekurencyjnymi implementacjami algorytmów, jak również rekurencjami w rozumieniu matematycznym. Do dzieła! UWAGA! Matematyczne rozwiązywanie rekurencji nie będzie obowiązywać na kartkówce.

1 Liczby Fibonacciego

Liczby Fibonacciego, to elementy ciagu Fibonacciego, zdefiniowanego jako

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2}, \quad \text{gdy } n > 1. \end{aligned}$$

Jest to definicja rekurencyjna, czyli taka, w której n-ty wyraz ciągu wyrażony jest za pomocą innych, najczęściej wcześniejszych jego wyrazów.

Jeśli wypiszemy kilka pierwszych wyrazów liczb Fibonacciego, zobaczymy

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

czyli ciąg, w którym każda następna liczba jest sumą dwóch poprzednich.

Zajmiemy się najpierw matematyczną analizą tego ciągu i potem przejdziemy do implementacji w Pythonie.

2 Matematyczne rozwiązywanie rekurencji

Zaprezentujemy najpierw technikę rozwiązywania takich rekurencji, czyli pozbywania się ich i odnajdowania wzorów ogólnych na n-ty wyraz ciągu.

Technika, którą wykorzystamy, opiera się na funkcji określonej wzorem

$$G(z) = \sum_{n=0}^{\infty} a_n z^n,$$

czyli szeregu potęgowym. Funkcja $G(z)$ powstała w ten sposób z ciągu a_n nazywa się jego funkcją tworzącą.

Jeśli jeszcze nie mieliście do czynienia z szeregiem potęgowym, można myśleć o nim jak o wielomianie stopnia ∞. Dwa wielomiany są sobie równe, wtedy i tylko wtedy, gdy mają te same współczynniki. Oznacza to, że dla każdego ciągu a_n istnieje tylko jedna funkcja $G(z)$ określona w pewnym tak zwanym „promieniu zbieżności". Na ten moment promień ten nie będzie nas interesował, ale więcej na ten temat dowiecie się na kursie z analizy matematycznej. Nam będzie potrzebna znajomość jedynie jednego szeregu tego typu, mianowicie szeregu geometrycznego

$$\sum_{n=0}^{\infty} a \cdot z^n = \frac{a}{1-z}.$$

Zacznijmy od zapisania funkcji tworzącej dla ciągu Fibonacciego

$$G(z) = \sum_{n=0}^{\infty} F_n z^n.$$

Teraz, ponieważ wzór na F_n jest inny dla $n = 0$, $n = 1$ oraz $n > 1$, wyciągnijmy dwa pierwsze elementy sumowania

$$G(z) = F_0 z^0 + F_1 z^1 + \sum_{n=2}^{\infty} F_n z^n.$$

Teraz podstawiamy definicję liczb Fibonacciego, czyli

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2}, \quad \text{gdy } n > 1. \end{aligned}$$

Mamy

$$G(z) = 0 \cdot 1 + 1 \cdot z + \sum_{n=2}^{\infty} (F_{n-1} + F_{n-2}) z^n = z + \sum_{n=2}^{\infty} F_{n-1} z^n + \sum_{n=2}^{\infty} F_{n-2} z^n.$$

Wyciągamy teraz z przed pierwszą sumą oraz z^2 przed drugą, aby zrównać wykładnik z indeksem ciągu. Następnie przesuwamy indeksy sumowania.

$$G(z) = z + z \sum_{n=2}^{\infty} F_{n-1} z^{n-1} + z^2 \sum_{n=2}^{\infty} F_{n-2} z^{n-2} = z + z \sum_{n=1}^{\infty} F_n z^n + z^2 \sum_{n=0}^{\infty} F_n z^n.$$

Zauważmy teraz, że ponieważ $F_0 = 0$, możemy zacząć pierwszą sumę od 0 zamiast od 1

$$G(z) = z + z \sum_{n=0}^{\infty} F_n z^n + z^2 \sum_{n=0}^{\infty} F_n z^n.$$

Przypomnijmy sobie, że

$$G(z) = \sum_{n=0}^{\infty} F_n z^n,$$

wobec czego

$$G(z) = z + zG(z) + z^2 G(z).$$

Rozwiązując oznacza to, że

$$G(z) \cdot (1 - z - z^2) = z,$$

czyli

$$G(z) = \frac{z}{1 - z - z^2},$$

Udało nam się znaleźć funkcję tworzącą ciągu Fibonacciego. Teraz, jeśli znajdziemy szereg potęgowy zbieżny do tej samej funkcji, jego wyrazy stojące przed znakiem sumowania będą ciągiem Fibonacciego.

Aby znaleźć taki ciąg, posłużymy się rozkładem funkcji wymiernej na ułamki proste — na potrzeby tego wykładu prześledzimy szczególny przypadek krok po kroku, jednak ogólna metoda rozkładania na ułamki proste będzie przedstawiona na kursie z algebry. Zacznijmy od znalezienie miejsc zerowych mianownika

$$1 - z - z^2 = 0,$$

stąd

$$\Delta = (-1)^2 - 4 \cdot (-1) \cdot 1 = 5,$$

czyli

$$z_1 = \frac{-(-1) - \sqrt{\Delta}}{2 \cdot (-1)} = \frac{\sqrt{5} - 1}{2},$$

$$z_2 = \frac{-(-1) + \sqrt{\Delta}}{2 \cdot (-1)} = \frac{-\sqrt{5} - 1}{2}.$$

Stąd

$$1 - z - z^2 = -(z - z_1)(z - z_2) = (z - z_1)(z_2 - z).$$

Na podstawie tych obliczeń, możemy stwierdzić, że

$$G(z) = \frac{z}{1 - z - z^2} = \frac{A}{z - z_1} + \frac{B}{z_2 - z}$$

dla pewnych stałych A i B. Sprowadźmy prawą stronę do wspólnego mianownika

$$\frac{A}{z - z_1} + \frac{B}{z_2 - z} = \frac{A(z_2 - z) + B(z - z_1)}{(z - z_1)(z_2 - z)} = \frac{A(z_2 - z) + B(z - z_1)}{1 - z - z^2}.$$

Wobec tego

$$G(z) = \frac{z}{1 - z - z^2} = \frac{A(z_2 - z) + B(z - z_1)}{1 - z - z^2},$$

przy czym porównując liczniki, otrzymujemy równanie

$$z = A(z_2 - z) + B(z - z_1).$$

Pozwoli to nam znaleźć wartości A i B. Wymnażamy prawą stronę i grupujemy tak, aby uzyskać wielomian zmiennej z

$$z = (B - A)z + Az_2 - Bz_1.$$

Zapisujemy układ równań

$$\begin{cases} B - A = 1, \\ Az_2 - Bz_1 = 0. \end{cases}$$

Z pierwszego równania wyznaczamy $B = A + 1$ i wstawiamy do drugiego równania.

$$Az_2 - Az_1 - z_1 = 0,$$

czyli

$$A(z_2 - z_1) = z_1$$

i ostatecznie

$$A = \frac{z_1}{z_2 - z_1} = \frac{\frac{\sqrt{5}-1}{2}}{\frac{-\sqrt{5}-1}{2} - \frac{\sqrt{5}-1}{2}} = \frac{\sqrt{5}-1}{-\sqrt{5}-1-\sqrt{5}+1} = \frac{1-\sqrt{5}}{2\sqrt{5}}.$$

Ponieważ $B = A + 1$, to

$$B = \frac{1-\sqrt{5}}{2\sqrt{5}} + 1 = \frac{1+\sqrt{5}}{2\sqrt{5}}.$$

Podsumowując, udało nam się zapisać

$$G(z) = \frac{z}{1-z-z^2} = \frac{A}{z-z_1} + \frac{B}{z_2-z} = \frac{\frac{1-\sqrt{5}}{2\sqrt{5}}}{z - \frac{\sqrt{5}-1}{2}} + \frac{\frac{1+\sqrt{5}}{2\sqrt{5}}}{\frac{-\sqrt{5}-1}{2} - z}.$$

Zacznijmy przekształcać prawą stronę tak, aby sprowadzić ułamki do postaci szeregu potęgowego. Zacznijmy od

$$\frac{\frac{1-\sqrt{5}}{2\sqrt{5}}}{z - \frac{\sqrt{5}-1}{2}}.$$

Podzielmy licznik i mianownik przez $-\frac{\sqrt{5}-1}{2}$

$$\frac{\frac{1-\sqrt{5}}{2\sqrt{5}}}{z - \frac{\sqrt{5}-1}{2}} = \frac{-\frac{2}{\sqrt{5}-1} \cdot \frac{1-\sqrt{5}}{2\sqrt{5}}}{-\frac{2}{\sqrt{5}-1} \cdot z + \frac{2}{\sqrt{5}-1} \cdot \frac{\sqrt{5}-1}{2}} = \frac{\frac{1}{\sqrt{5}}}{1 - \left(\frac{2}{\sqrt{5}-1} \cdot z\right)}.$$

Korzystając z

$$\sum_{n=0}^{\infty} a \cdot z^n = \frac{a}{1-z}.$$

mamy

$$\frac{\frac{1-\sqrt{5}}{2\sqrt{5}}}{z - \frac{\sqrt{5}-1}{2}} = \frac{\frac{1}{\sqrt{5}}}{1 - \left(\frac{2}{\sqrt{5}-1} \cdot z\right)} = \sum_{n=0}^{\infty} \frac{1}{\sqrt{5}} \left(\frac{2}{\sqrt{5}-1} \cdot z\right)^n = \sum_{n=0}^{\infty} \frac{1}{\sqrt{5}} \left(\frac{2}{\sqrt{5}-1}\right)^n z^n.$$

Przejdźmy do rozpisania wyrażenia

$$\frac{\frac{1+\sqrt{5}}{2\sqrt{5}}}{\frac{-\sqrt{5}-1}{2} - z}.$$

Podzielmy licznik i mianownik przez $\frac{-\sqrt{5}-1}{2}$

$$\frac{\frac{1+\sqrt{5}}{2\sqrt{5}}}{\frac{-\sqrt{5}-1}{2}-z} = \frac{\frac{2}{-\sqrt{5}-1}\cdot\frac{1+\sqrt{5}}{2\sqrt{5}}}{\frac{2}{-\sqrt{5}-1}\cdot\frac{-\sqrt{5}-1}{2}-\frac{2}{-\sqrt{5}-1}\cdot z} = \frac{-\frac{1}{\sqrt{5}}}{1-\left(\frac{2}{-\sqrt{5}-1}\cdot z\right)}.$$

Wykorzystując wzór na sumę szeregu geometrycznego

$$\frac{\frac{1+\sqrt{5}}{2\sqrt{5}}}{\frac{-\sqrt{5}-1}{2}-z} = \frac{-\frac{1}{\sqrt{5}}}{1-\left(\frac{2}{-\sqrt{5}-1}\cdot z\right)} = \sum_{n=0}^{\infty} -\frac{1}{\sqrt{5}}\left(\frac{2}{-\sqrt{5}-1}\cdot z\right)^n = \sum_{n=0}^{\infty} -\frac{1}{\sqrt{5}}\left(\frac{2}{-\sqrt{5}-1}\right)^n z^n.$$

Wykorzystując te postaci mamy

$$\begin{aligned} G(z) &= \frac{A}{z-z_1}+\frac{B}{z_2-z} = \frac{\frac{1-\sqrt{5}}{2\sqrt{5}}}{z-\frac{\sqrt{5}-1}{2}}+\frac{\frac{1+\sqrt{5}}{2\sqrt{5}}}{\frac{-\sqrt{5}-1}{2}-z} = \\ &= \sum_{n=0}^{\infty}\frac{1}{\sqrt{5}}\left(\frac{2}{\sqrt{5}-1}\right)^n z^n + \sum_{n=0}^{\infty} -\frac{1}{\sqrt{5}}\left(\frac{2}{-\sqrt{5}-1}\right)^n z^n = \\ &= \sum_{n=0}^{\infty}\left[\frac{1}{\sqrt{5}}\left(\frac{2}{\sqrt{5}-1}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{2}{-\sqrt{5}-1}\right)^n\right] z^n. \end{aligned}$$

Przypomnijmy, że wyszliśmy od

$$G(z) = \sum_{n=0}^{\infty} F_n z^n,$$

więc porównując szeregi mamy

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{2}{\sqrt{5}-1}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{2}{-\sqrt{5}-1}\right)^n.$$

Zauważmy, że wykorzystując złotą liczbę

$$\varphi = \frac{1+\sqrt{5}}{2},$$

możemy wzór na F_n zapisać jako

$$F_n = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}},$$

ponieważ

$$\frac{2}{\sqrt{5}-1} = \frac{2}{\sqrt{5}-1}\cdot\frac{1+\sqrt{5}}{1+\sqrt{5}} = \frac{2(1+\sqrt{5})}{5-1} = \frac{1+\sqrt{5}}{2} = \varphi$$

oraz

$$\frac{2}{-\sqrt{5}-1} = \frac{2}{-\sqrt{5}-1}\cdot\frac{\sqrt{5}-1}{\sqrt{5}-1} = \frac{2(\sqrt{5}-1)}{1-5} = \frac{1-\sqrt{5}}{2} = 1-\frac{1+\sqrt{5}}{2} = 1-\varphi.$$

3 Nierekurencyjna postać liczb Fibonacciego

W poprzedniej sekcji rozwiązaliśmy rekurencję definiującą ciąg Fibonacciego

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2}, \quad \text{gdy } n > 1. \end{aligned}$$

jako

$$F_n = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}},$$

gdzie φ to złota liczba, czyli

$$\varphi = \frac{1+\sqrt{5}}{2}.$$

Jest to tak zwany wzór Bineta.

Niestety po wpisaniu wzoru do Pythona przez błędy zaokrągleń typu zmiennoprzecinkowego i niewymierność złotej liczby, wartości liczb Fibonacciego wyznaczonych za pomocą wzoru Bineta nie będą liczbami całkowitymi.

Zauważmy, że

$$\left|\frac{(1-\varphi)^n}{\sqrt{5}}\right| < \frac{1}{2}.$$

Jak to pokazać? Zauważamy, że $|1-\varphi| < 1$, więc $\left|\frac{(1-\varphi)^n}{\sqrt{5}}\right|$ maleje ze względu na n. Wobec tego, wystarczy, aby nierówność była prawdziwa dla $n = 0$. Mamy

$$\frac{1}{\sqrt{5}} < \frac{1}{\sqrt{4}} = \frac{1}{2},$$

co w połączeniu z obserwacją o monotoniczności ze względu na n kończy uzasadnienie nierówności. Stąd

$$F_n = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}} = \frac{\varphi^n}{\sqrt{5}} - \frac{(1-\varphi)^n}{\sqrt{5}} = \frac{\varphi^n}{\sqrt{5}} - \frac{1}{2} + \varepsilon_n,$$

gdzie

$$\varepsilon_n = \frac{1}{2} - \frac{(1-\varphi)^n}{\sqrt{5}}$$

i ponieważ

$$\frac{(1-\varphi)^n}{\sqrt{5}} \in \left(-\frac{1}{2}, \frac{1}{2}\right),$$

to

$$\varepsilon_n \in (0, 1).$$

Ponieważ F_n jest liczbą całkowitą i $\varepsilon_n \in (0,1)$, to

$$F_n - 1 < \frac{\varphi^n}{\sqrt{5}} - \frac{1}{2} < F_n$$

co pozwala zapisać wzór

$$F_n = \left\lceil \frac{\varphi^n}{\sqrt{5}} - \frac{1}{2} \right\rceil = \left\lfloor \frac{\varphi^n}{\sqrt{5}} - \frac{1}{2} \right\rfloor + 1 = \left\lfloor \frac{\varphi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor.$$

Wzór ten po wprowadzeniu do Pythona zapiszemy następująco

```
import math

def fib_wzór(n):
    return int(((math.sqrt(5)+1)/2)**n/math.sqrt(5) + 0.5)
```

Jest to bardzo szybki algorytm, niestety daje prawidłowe rezultaty jedynie dla $n \leqslant 70$, ponieważ dla większych liczb zaokrąglenia wynikające z zastosowania liczb zmiennoprzecinkowych akumulują się i są w sumie większe niż 1, zatem zaokrąglanie w dół przestaje działać[1]. Dla większych wartości n musimy poszukać innego rozwiązania.

4 Implementacja rekurencyjna

W programowaniu w Pythonie wewnątrz definicji funkcji możemy z niej od razu skorzystać. Technika ta, tak samo jak w matematyce, nazywa się rekurencją. Wzór

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2}, \quad \text{gdy } n > 1. \end{aligned}$$

implementujemy jako

```
def fib_rekurencja(n):
    if n < 2:
        return n
    return fib_rekurencja(n-1) + fib_rekurencja(n-2)
```

Implementacja ta jest poprawna i wydaje się najbardziej naturalną implementacją wzoru zadanego rekurencyjnie. Jesteśmy jedynie ograniczeni wielkością stosu, więc powinniśmy być w stanie policzyć liczby Fibonacciego aż do 3000 wyrazu ciągu (dla domyślnej wielkości stosu w Jupyter Notebook).

Niestety, jeśli zaczniemy wyliczać czas działania algorytmu, zobaczymy następujące rezultaty.

n	czas
0	$151ns$
1	$151ns$
5	$1.64\mu s$
15	$219\mu s$
20	$2.45ms$
33	$1.2s$

[1] Liczba 70 została znaleziona eksperymentalnie przez porównanie z inną implementacją w pętli.

Czas oczywiście może być nieznacznie inny na każdym komputerze. Widać, że coś jest nie tak — czas wykonania algorytmu rośnie bardzo szybko.

4.1 Analiza czasu działania algorytmu

Zastanówmy się, ile razy wywoływana jest funkcja `fib_rekurencja`, gdy ktoś wpisze `fib_rekurencja(n)`. Jeśli $n < 2$, będzie tylko jedno wywołanie (to, które wykonał użytkownik). Jeśli $n \geqslant 2$, będzie to 1 plus liczba wywołań dla $n-1$ oraz $n-2$. Wobec tego liczbę wywołań funkcji możemy zapisać wzorem rekurencyjnym. Oznaczmy ją N_n.

$$\begin{aligned} N_0 &= 1, \\ N_1 &= 1, \\ N_n &= 1 + N_{n-1} + N_{n-2}, \quad \text{gdy } n > 1. \end{aligned}$$

Wzór ten przypomina definicję liczb Fibonacciego. Możemy matematycznie rozwiązać tę rekurencję w sposób analogiczny jak dla liczb Fibonacciego i uzyskać postać

$$N_n = 2F_{n+1} - 1.$$

Ponieważ

$$F_n \approx \frac{\varphi^n}{\sqrt{5}},$$

otrzymujemy

$$N_n = 2F_{n+1} - 1 \approx \frac{2\varphi^{n+1}}{\sqrt{5}} \propto \varphi^n.$$

Oznacza to, że N_n jest w przybliżeniu proporcjonalne do φ^n, przy czym wartość ta rośnie bardzo szybko.

Problem polega na tym, że niektóre wartości funkcji `fib_rekurencja` obliczane są wiele razy. Na przykład, aby policzyć `fib_rekurencja(4)` potrzeba wartości w 3 i 2. Do obliczenia `fib_rekurencja(3)`, potrzeba wartości w 2 i 1. Już na tym etapie, `fib_rekurencja(2)` obliczana jest dwukrotnie. Problem możemy rozwiązać, stosując zapamiętywanie wyników częściowych. Można na przykład zastosować dekorator `lru_cache` z biblioteki `functools`.

```
from functools import lru_cache

@lru_cache
def fib_rekurencja(n):
    if n < 2:
        return n
    return fib_rekurencja(n-1) + fib_rekurencja(n-2)
```

Zastosowanie tego dekoratora rozwiązuje nasz problem i powoduje, że wyniki składowe są zapamiętywane i nie są liczone dwukrotnie.

Niestety, wciąż nie rozwiązuje to problemu rosnącego stosu, co uniemożliwia nam obliczenie dużych liczb Fibonacciego.

5 Implementacja iteracyjna

Pamiętając o tym, że każda kolejna liczba Fibonacciego to suma dwóch poprzednich, możemy zaimplementować algorytm iteracyjny obliczający liczby F_n w kolejności od najmniejszej aż trafimy na poszukiwaną.

```
def fib_iteracyjnie(n):
    if n <= 1:
        return n
    a, b = 0, 1
    while n > 1:
        a, b = b, a+b
        n -= 1
    return b
```

Zachęcam, aby przeanalizować działanie tego programu na stronie PythonTutor[2]. Jako ćwiczenie proponuję również zmierzyć czas wykonania algorytmu i porównać go z różnymi innymi implementacjami.

6 Połączenie algorytmów

Podczas analizy czasu okaże się, że najszybszy jest algorytm `fib_wzór`, zaraz potem `fib_iteracyjnie` i na końcu `fib_rekurencja`. Niestety, `fib_wzór` nie działa dla wszystkich n. Co możemy zrobić? Możemy połączyć dwa najszybsze algorytmy w jeden jeszcze lepszy.

```
import math


def fib(n):
    golden = (math.sqrt(5)+1)/2
    if n <= 70:
        return int(golden**n/math.sqrt(5) + 0.5)
    golden69 = golden**69/math.sqrt(5)
    a, b = int(golden69+0.5), int(golden69*golden+0.5)
    while n > 70:
        a, b = b, a+b
        n -= 1
    return b
```

Algorytm ten dla $n \leqslant 70$ wykorzystuje obliczony wzór, natomiast dla większych wartości korzysta ze wzoru iteracyjnego, zaczynając od wyliczonych wzorem F_{69} oraz F_{70} — dwóch największych liczb Fibonacciego dla, których działa wzór oparty o wzór Bineta.

[2] http://pythontutor.com/visualize.html

Dla liczb $n \leqslant 70$ wzór ten jest tak samo szybki jak wersja `fib_wzór`, natomiast dla większych n, będzie szybszy niż `fib_iteracyjnie`, ponieważ nie musi wykonywać 70 przejść pętli. Na przykład F_{140} jest dwukrotnie szybsze w implementacji `fib` niż `fib_iteracyjnie`, a wciąż otrzymujemy poprawny wynik dla każdego n.

Algorytmy takie, które posiadają w sobie kilka różnych podejść do rozwiązania problemu i dobierają podejście zależnie od argumentów, nazywamy algorytmami adaptacyjnymi.

7 Jeszcze lepszy algorytm

Można pomyśleć, że wersje iteracyjne są szybsze od rekurencyjnych, ponieważ tak nam wyszło tym razem. Warto jednak podkreślić, że uzyskany algorytm jest najszybszym z rozważanych, ale nie najszybszym znanym. O równaniu rekurencyjnym prawdziwym dla liczb Fibonacciego można przeczytać w artykule „In honour of Fibonacci" autorstwa E. W. Dijkstry[3]. Połączenie implementacji wzorem dla $n \leqslant 70$ oraz wzoru z artykułu zaimplementowanego rekurencyjnie z zapamiętywaniem częściowych wyników pozwala na uzyskanie błyskawicznej implementacji (choć wciąż nie gwarantujemy, że nie da się lepiej!).

```
import math
from functools import lru_cache

@lru_cache
def fib2(n):
    if n <= 70:
        return int(((math.sqrt(5)+1)/2)**n/math.sqrt(5) + 0.5)
    else:
        j = (n-1)//2
        fib_j = fib2(j)
        fib_j1 = fib2(j+1)
        if n%2 == 0:
            return (2*fib_j+fib_j1)*fib_j1
        else:
            return fib_j*fib_j + fib_j1*fib_j1
```

Spróbuj porównać tę implementację z innymi do tej pory przeanalizowanymi. Sprawdź, że daje te same wyniki co pozostałe przykładowe implementacje i zmierz czas wykonania algorytmu. Prześledź zapiski w artykule i zwróć uwagę na różnice pomiędzy tamtą implementacją a przedstawioną tutaj, w szczególności na numerację liczb Fibonacciego.

8 Podsumowanie

W niniejszym dokumencie prześledziliśmy proces powstawania możliwej implementacji rekurencyjnego wzoru matematycznego. Pokazaliśmy również, jak badanie własności

[3] https://www.cs.utexas.edu/users/EWD/ewd06xx/EWD654.PDF

algorytmów pozwala nam wybrać najszybszy program zależnie do sytuacji oraz jak kreatywnie wykorzystać kilka szybkich i dopracowanych implementacji, by połączyć je w jedną jeszcze lepszą. Pokazuje to, że nakład pracy włożony w analizę i przygotowanie algorytmu, zwraca się nam, gdy uzyskamy coś znacznie szybszego, niż naiwna implementacja.