Ciąg Fibonacciego w języku Java

Źródło: Jason Leung, domena publiczna.

Ciąg Fibonacciego to ciąg, w którym każdy kolejny element, z wyjątkiem dwóch początkowych, jest sumą dwóch poprzednich elementów. Spotykamy go nie tylko w matematyce, ale również w biologii, literaturze i w muzyce. W tym e-materiale dowiemy się, jak w języku Java napisać program, który wygeneruje kolejne elementy ciągu Fibonacciego.

Więcej informacji o ciągu Fibonacciego znajdziesz w e-materiałach:

- Ciąg Fibonacciego,
- Ciąg Fibonacciego w języku C++,
- Ciąg Fibonacciego w języku Python.

Więcej zadań? Zajrzyj do e-materiału Ciąg Fibonacciego – zadania maturalne.

O tym, jak zagadnienie rekurencji wyjaśnia matematyka, przeczytasz w e-materiałach:

- Ciąg określony rekurencyjnie,
- Ciąg geometryczny określony rekurencyjnie,
- Wzór ogólny ciągu określonego rekurencyjnie,
- Ciąg arytmetyczny określony wzorem rekurencyjnym.

Twoje cele

- Przeanalizujesz rekurencyjny algorytm generujący kolejne elementy ciągu Fibonacciego.
- Napiszesz programy wyznaczające elementy ciągu Fibonacciego w sposób rekurencyjny oraz iteracyjny.
- Rozwiążesz kilka zadań związanych z tematem e-materiału.

Przeczytaj

Na czym polega rekurencja?

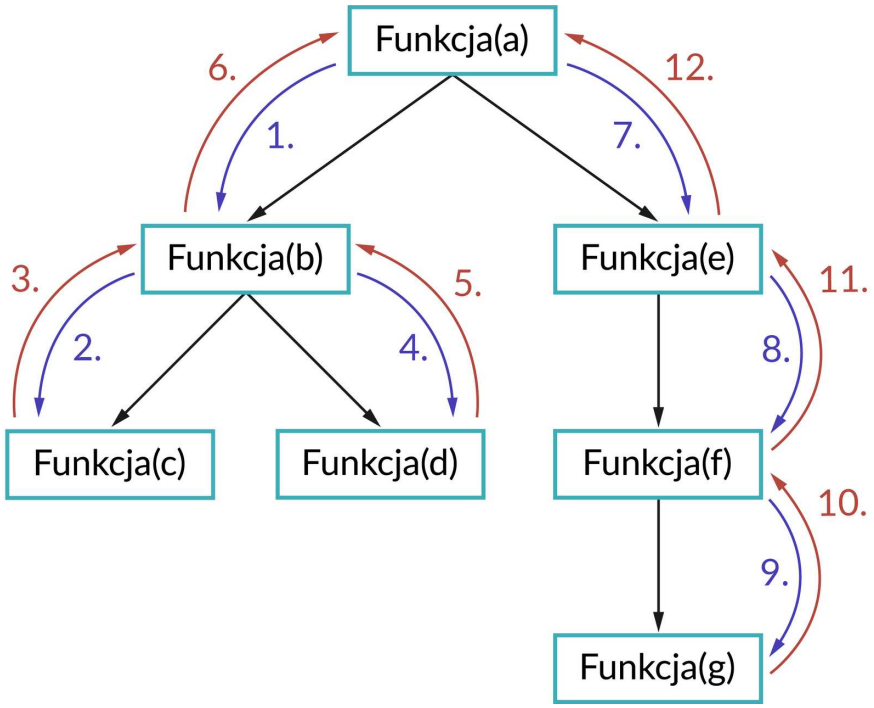
Istotą rekurencji jest wykorzystanie funkcji, która – aby rozwiązać pewien problem – wywołuje samą siebie, aż do momentu, gdy zadanie zostanie wykonane. Funkcja ta może być typu `void`, gdy wywoływana funkcja nie zwraca wartości do funkcji wywołującej, lecz np. wypisuje ją do konsoli. Najczęściej jednak funkcja rekurencyjna oblicza oraz zwraca wartość.

Aby mechanizm rekurencji działał poprawnie, potrzebny jest przynajmniej jeden warunek, po spełnieniu którego funkcja nie zostanie wywołana po raz kolejny, lecz zakończy działanie (a także zwróci wartość – o ile miała ją zwracać). Jeżeli taki warunek się nie pojawi, funkcja będzie wywoływać się w nieskończoność – może to spowodować zawieszenie programu – jak w przypadku nieskończonej pętli – lub przepełnienie stosu. Nie otrzymamy wówczas rozwiązania problemu.

Z każdym kolejnym wywołaniem funkcji rekurencyjnej przynajmniej jeden z jej argumentów musi zmienić wartość. Zmiany argumentów muszą prowadzić do przypadku bazowego. Wartości argumentów nie mogą powtórzyć się w następnych wywołaniach, z wyjątkiem argumentów, których wartości przerywają dany ciąg wywołań rekurencyjnych. Wartości argumentów stanowią wtedy przypadek podstawowy rekurencji. Jeżeli złamiemy chociaż jedną z przedstawionych zasad, funkcja rekurencyjna będzie wywoływać się bez końca. Kolejne wywołanie zostaje zapisane na stosie, więc gdy funkcja będzie uruchamiana w nieskończoność, w pewnym momencie stos przepełni się, a program zakończy działanie.

Każde rekurencyjne wywołanie funkcji zatrzymuje wykonywanie funkcji bieżącej. Czeka ona, aż funkcja przez nią wywołana zakończy działanie i zwróci wartość. Przy wywoływaniu kolejnych funkcji rekurencyjnych poprzednie funkcje także zatrzymują się i czekają, aż funkcje przez nie wywołane zwrócą obliczoną wartość.

Opisany proces można przedstawić graficznie przy użyciu drzewa wywołań rekurencyjnych.



Z przedstawionej grafiki wynika, że pierwszym wywołaniem funkcji rekurencyjnej `Funkcja` było: `Funkcja(a)`. Ta z kolei w pierwszej kolejności wywołała rekurencyjnie `Funkcja(b)`. A zatem wywołanie `Funkcja(a)` czeka, aż wywołana `Funkcja(b)` zwróci wartość.

Kolejnym krokiem było wywołanie `Funkcja(c)`. Wywołanie funkcji z takim argumentem nie spowodowało kolejnego wywołania rekurencyjnego, lecz zwrócenie wartości. Zobrazowane to zostało przy użyciu czerwonej strzałki z oznaczeniem 3. Jednak `Funkcja(b)` wywołała nie jedno, lecz dwa wykonania rekurencyjne. Drugim jest `Funkcja(d)`, które podobnie jak `Funkcja(c)`, zamiast wywoływać dalsze wykonania rekurencyjne, zwróciła wartość.

Kolejny krok przedstawia czerwona strzałka z numerem 6. Wywołanie `Funkcja(b)` zwraca wartość do `Funkcja(a)`. Następnym krokiem jest wywołanie kolejno: `Funkcja(e)` → `Funkcja(f)` → `Funkcja(g)`, po czym ostatnie wywołanie zwraca wartość do poprzedniego i w ten sposób dochodzimy do pierwszego wywołania `Funkcja(a)`, które finalnie zwraca wartość.

Rekurencja – przykład

Zanim przejdziemy do ciągu Fibonacciego, napiszmy prosty program wykorzystujący mechanizm rekurencji. Program ten zwróci sumę wszystkich liczb naturalnych mniejszych lub równych liczbie podanej jako argument.

Rozpoczynamy od zdeklarowania funkcji.

```
1 public static int sumaLiczbNaturalnych(int n) {
```

```
2
3 }
```

Ponieważ funkcja ma obliczyć sumę liczb naturalnych, typem zwracanych danych jest `int`. Parametr `n` będzie górną granicą składników sumy.

Zaczniemy od zapisania warunku, po spełnieniu którego funkcja zakończy działanie i zwróci wartość. Warunek będzie dotyczył najmniejszej możliwej wartości parametru `n`. Zakładamy, że liczby naturalne zaczynają się od 1. Ile wynosi suma liczb naturalnych mniejszych lub równych 1? Jest to 1. Skonstruujmy więc instrukcję warunkową, która będzie odpowiedzialna za działanie programu w przypadku, gdy wartość `n` wynosi 1:

```
1 public static int sumaLiczbNaturalnych(int n) {
2     if (n == 1) {
3         return 1;
4     }
5 }
```

Kolejnym i ostatnim krokiem jest rekurencyjne wywołanie funkcji.

```
1 public static int sumaLiczbNaturalnych(int n) {
2     if (n == 1) {
3         return 1;
4     }
5
6     return n + sumaLiczbNaturalnych(n - 1);
7 }
```

Ponieważ chcemy uzyskać sumę liczb naturalnych, a różnica między kolejnymi liczbami naturalnymi zawsze wynosi 1, za każdym razem, gdy wywołujemy kolejną funkcję, odejmujemy od jej argumentu wartość 1. Do wyniku tego wywołania dodajemy wartość aktualnego argumentu i ta suma zostaje przekazana funkcji, która wywołała bieżącą funkcję.

Aby lepiej zrozumieć działanie programu, sprawdźmy, jak się on zachowa, gdy jako argument podamy wartość 3.

```
1 public static int sumaLiczbNaturalnych(int n) {
2     // n = 3
3
```

```
    if (3 == 1) {
        return 1;
    }

    return 3 + sumaLiczbNaturalnych(2);
}
```

Sprawdzamy, czy 3 jest równe 1. Nie jest to prawdą, więc pomijamy instrukcję warunkową i przechodzimy do wywołania rekurencyjnego. Liczbę 3 dodajemy do wartości zwróconej przez `sumaLiczbNaturalnych(2)`. Funkcja przerywa działanie i przechodzi do funkcji, którą sama wywołała, czyli `sumaLiczbNaturalnych(2)`.

```
public static int sumaLiczbNaturalnych(int n) {
    // n = 2

    if (2 == 1) {
        return 1;
    }

    return 2 + sumaLiczbNaturalnych(1);
}
```

Jak wiemy, 2 również nie jest równe 1. Pomijamy warunek i ponownie zatrzymujemy wykonywanie tej funkcji. Zwróćmy uwagę na funkcję, która została wywołana, czyli `sumaLiczbNaturalnych(1)`.

```
public static int sumaLiczbNaturalnych(int n) {
    // n = 1

    if (1 == 1) {
        return 1;
    }
}
```

W tym przypadku instrukcja warunkowa nie zostanie pominięta. Skoro `1 == 1`, nie wywołujemy już funkcji `sumaLiczbNaturalnych()`, lecz zwracamy wartość 1. Przekazujemy ją funkcji, która wywołała bieżącą funkcję.

```
public static int sumaLiczbNaturalnych(int n) {
```

```
    // n = 2

    if (2 == 1) {
        return 1;
    }

    return 2 + 1;
}
```

Ta z kolei otrzymała wartość, na którą czekała. Może więc dodać ją do swojego argumentu i zwrócić obliczony wynik funkcji, która wywołała tę wartość.

```
public static int sumaLiczbNaturalnych(int n) {
    // n = 3

    if (3 == 1) {
        return 1;
    }

    return 3 + 3;
}
```

Wywołana pierwotnie funkcja zwraca wartość 6, która jest sumą wszystkich liczb naturalnych mniejszych lub równych 3.

Ciąg Fibonacciego

Definicja ciągu Fibonacciego wygląda następująco: początkowymi elementami ciągu są 0 oraz 1, natomiast każdy kolejny element jest sumą dwóch poprzednich. Trzecim elementem ciągu Fibonacciego będzie zatem $0 + 1 = 1$, czwartym $1 + 1 = 2$, piątym $1 + 2 = 3$ itd. Indeksy elementów zaczynają się od 0, podobnie jak indeksy tablicy w języku Java. Pierwszy element – o wartości 0 – ma indeks `n = 0`.

Rekurencja

Problem 1

Napisz w języku Java program, który zwróci wyraz ciągu Fibonacciego o indeksie n. Zastosuj mechanizm rekurencyjny. Przetestuj jego działanie dla n = 3.

Ciąg Fibonacciego to ciąg liczb, w którym pierwszy wyraz jest równy 0, drugi jest równy 1, a każdy następny jest sumą dwóch poprzednich. Ciąg ten można przedstawić za pomocą rekurencyjnego wzoru funkcji $F(n)$, obliczającej $n+1$ element ciągu Fibonacciego:

$$F_n = \begin{cases} 0 & \text{dla } n = 0 \\ 1 & \text{dla } n = 1 \\ F_{n-1} + F_{n-2} & \text{dla } n > 1 \end{cases}$$

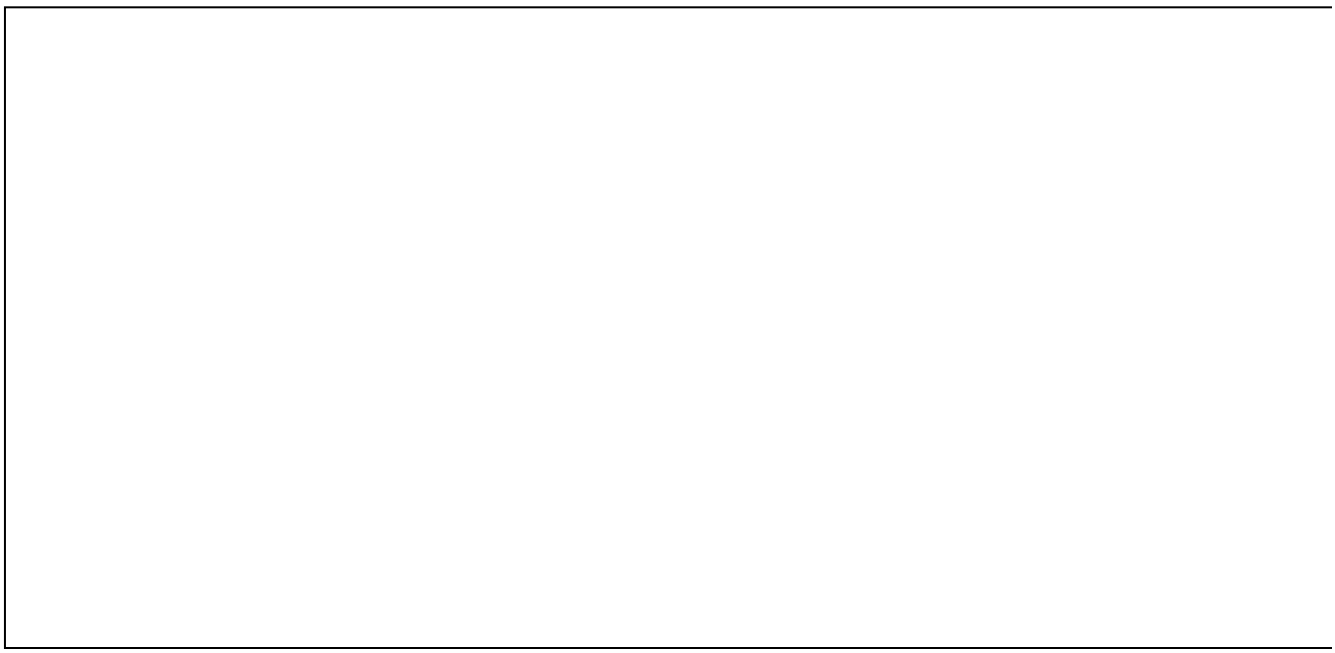
Specyfikacja:

Dane:

- n – indeks elementu ciągu; liczba naturalna

Wynik:

Program oblicza wyraz ciągu Fibonacciego o indeksie n.



Polecenie 1

Porównaj swoje rozwiązanie z prezentacją.

1

Napiszemy program, który zwróci element ciągu Fibonacciego o indeksie n. Zastosujemy mechanizm rekurencyjny. Zaczniemy od utworzenia funkcji.

2

Ponieważ elementami ciągu Fibonacciego są liczby naturalne, zastosujemy typ całkowitoliczbowy. Zwrócimy typ `long`,

ponieważ ciąg Fibonacciego jest ciągiem szybko rosnącym. Parametr n funkcji `fibonacci` będzie indeksem elementu ciągu, który ma być obliczony.

3

Następnym krokiem jest zapisanie warunku, dla którego nowa instancja funkcji nie zostanie uruchomiona. W przypadku ciągu Fibonacciego trzeba uwzględnić dwa warunki. Wiemy bowiem, że pierwszym elementem ciągu (o indeksie 0) jest 0, a drugim elementem (o indeksie 1) jest 1.

4

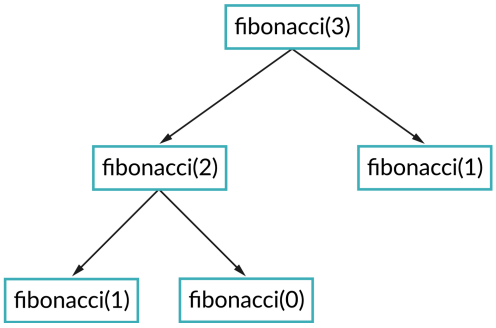
Zapiszmy obydwa warunki.

5

Ostatnim etapem jest rekurencyjne wywołanie funkcji. Wykorzystamy fakt, że kolejny element jest sumą dwóch poprzednich.

6

Aby sprawdzić, czy program działa poprawnie, użyjemy go do wyznaczenia elementu o indeksie 3. Wywołujemy funkcję `fibonacci(3)`.



7

Wywołanie `fibonacci(3)` może zostać przedstawione na drzewie wywołań rekurencyjnych. Zwróćmy uwagę, że wywołania `fibonacci(1)` oraz `fibonacci(0)` są wywołaniami elementarnymi, które nie powodują kolejnych wywołań rekurencyjnych, lecz zwracają konkretną wartość.

8

Żaden z warunków nie został spełniony, więc wywołana zostaje funkcja `fibonacci(2)` w celu obliczenia pierwszej części sumy.

9

Co się stanie z wywołaniem `fibonacci(2)`? Sprawdźmy.

10

Zostaje wywołana kolejna instancja funkcji `fibonacci()`, tym razem z argumentem równym 1. Jak już wiemy, jest to wywołanie elementarne (spełniony zostanie drugi warunek w instrukcji warunkowej), zwróci więc wartość 1. Następnie zostanie wywołana funkcja `fibonacci(0)`, która zwróci wartość 0.

Możemy przekazać te wartości funkcji, która wywołała obie instancje.

11

Natomiast ta funkcja zwróci wartość `1 + 0` funkcji, która ją wywołała. Zakończyło się wywołanie `fibonacci(2)`, zatem przechodzimy do drugiej części sumy, czyli do wywołania `fibonacci(1)`, które to jest wywołaniem elementarnym i zwraca wartość 1, czyli:

12

Wartość `1 + 1` zostanie zwrócona jako odpowiedź. Zatem elementem ciągu Fibonacciego o indeksie 3 jest 2.

13

Oto kod całego programu:

Iteracja

Napiszemy program, który zwróci element ciągu Fibonacciego o indeksie `n`. Zastosujemy iterację. Zaczniemy ponownie od utworzenia funkcji.

```
1 public static long iterFibonacci(int n) {
2
```

```
}
```

Wybieramy typ `long`, ponieważ wartości kolejnych wyrazów ciągu Fibonacciego rosną wykładniczo.

Parametr `n` funkcji będzie indeksem elementu ciągu, który ma być obliczony.

Zapisujemy warunki, po których spełnieniu zostanie zatrzymane działanie funkcji i zwróci ona wartość. Wiemy, że pierwszy wyraz ciągu Fibonacciego – zatem wyraz o indeksie 0 – ma wartość 0. Natomiast drugi wyraz – czyli wyraz o indeksie 1 – wynosi 1. Zapiszemy odpowiednie instrukcje warunkowe.

```
public static long iterFibonacci(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    }
}
```

Musimy brać również pod uwagę przypadek, gdy wartość parametru n jest większa niż 1. W takiej sytuacji każdy kolejny element jest sumą dwóch poprzednich. Dla dalszego działania funkcji potrzebujemy zmiennych pomocniczych. Inicjalizujemy trzy zmienne pomocnicze: `a`, `b` (oznaczające wartości dwóch początkowych wyrazów ciągu) oraz `i` (indeks kolejnego elementu ciągu Fibonacciego). Przypisujemy im odpowiednio wartości 0, 1 i 2.

```
public static long iterFibonacci(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    }

long a = 0;
long b = 1;
int i = 2;

}
```

Zapisujemy pętlę while, która będzie wykonywać się tak długo, dopóki wartość zmiennej `i` jest mniejsza od wartości zmiennej `n` lub jej równa.

```
public static long iterFibonacci(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    }

long a = 0;
long b = 1;
int i = 2;

while (i <= n) {

    }

}
```

W każdym obiegu pętli `while` najpierw obliczamy `i`-ty wyraz ciągu, czyli sumę dwóch poprzednich wyrazów ciągu. Wynik przypisujemy do zmiennej `suma`. Następnie zmiennej `a` przypisujemy wartość zmiennej `b`, a zmiennej `b` wartość zmiennej `suma`. Na końcu zwiększamy wartość zmiennej `i` o 1.

```
public static long iterFibonacci(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    }

long a = 0;
long b = 1;
int i = 2;

while (i <= n) {
    long suma = a + b;
    a = b;
    b = suma;
```

```
    i = i + 1;

    }

}
```

Po zakończeniu pętli wartość zmiennej `b` jest zwracana jako wynik działania funkcji.

```
public static long iterFibonacci(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    }

long a = 0;
long b = 1;
int i = 2;

while (i <= n) {
    long suma = a + b;
    a = b;
    b = suma;
    i = i + 1;

    }

return b;

}
```

Oto cały kod programu:

```
public class Main {
    public static void main(String args[]) {
        System.out.println(iterFibonacci(3));
    }

    public static long iterFibonacci(int n) {
        if (n == 0) {
```

```
8             return 0;
9         } else if (n == 1) {
10            return 1;
11        }
12
13        long a = 0;
14        long b = 1;
15        int i = 2;
16
17        while (i <= n) {
18            long suma = a + b;
19            a = b;
20            b = suma;
21            i = i + 1;
22        }
23        return b;
24    }
25}
```

Słownik

argument funkcji

element składni w określonym języku programowania, który w wyniku wywołania podprogramu zostaje utożsamiony (skojarzony) z określonym parametrem podprogramu
funkcja typu void

funkcja, która nie zwraca żadnej wartości
instrukcja warunkowa

element języka programowania sterujący działaniem programu; pozwala wykonać różne instrukcje w zależności od tego, czy zdefiniowane przez programistę wyrażenie logiczne jest prawdziwe czy nieprawdziwe
nieskończona pętla

pętla, która nigdy nie spełnia warunku zakończenia; pętla działająca w nieskończoność
parametr funkcji

element składni w określonym języku programowania; umożliwia komunikację pomiędzy podprogramem (funkcją) wywołanym a programem wywołującym; parametry określa się w nagłówku podprogramu (przy jego definicji)
rekurencja

> sytuacja, w której funkcja wywołuje samą siebie, aż do napotkania przypadku podstawowego

Gra edukacyjna

Polecenie 1

Sprawdź swoją wiedzę, biorąc udział w grze.

Test

Sprawdź swoją wiedzę o ciągu Fibonacciego w języku Java, biorąc udział w grze.

Poziom trudności:

łatwy

Limit czasu:

7 min

Twój ostatni wynik:

-

Uruchom

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Napisz program, który wypisze wszystkie wyrazy ciągu Fibonacciego mniejsze od liczby n w kolejności niemalejącej. Dla dwóch pierwszych wyrazów należy przyjąć, że są równe odpowiednio `0` i 1. Swój program przetestuj dla `n = 100`.

Specyfikacja:

Dane:

- n – liczba naturalna, od której wypisane wyrazy mają być mniejsze

Wynik:

Program wypisuje wszystkie wyrazy ciągu Fibonacciego mniejsze od n, w kolejności niemalejącej, każdy wyraz w nowej linii.

Ćwiczenie 2

Napisz program, który przedstawi liczbę n jako sumę dwóch wyrazów ciągu Fibonacciego lub wypisze komunikat BRAK w przypadku braku rozwiązania. Rozwiązanie, jeśli istnieje, należy wypisać w postaci dwóch liczb oddzielonych spacją, podanych w kolejności niemalejącej. Dla dwóch pierwszych wyrazów należy przyjąć, że są równe odpowiednio 0 i 1. Przetestuj swój program dla n = 42.

Specyfikacja:

Dane:

- n – liczba naturalna, której rozkład należy obliczyć; $n \geq 2$

Wynik:

Program wyświetla dwa elementy ciągu Fibonacciego, oddzielone spacją w kolejności niemalejącej, które sumują się do n lub komunikat BRAK w przypadku braku rozwiązania.

Ćwiczenie 3

Napisz program, który obliczy liczbę wywołań rekurencyjnych funkcji `fibonacci()` dla obliczenia elementu ciągu Fibonacciego o indeksie n. Uwzględniamy wszystkie wywołania. Przetestuj działanie programu dla elementu ciągu o indeksie 11.

Specyfikacja:

Dane:

- n – indeks elementu ciągu Fibonacciego (indeksy liczone od 0); liczba naturalna

Wynik:

Program wyświetla liczbę wywołań rekurencyjnych danej funkcji dla obliczenia elementu ciągu Fibonacciego o indeksie n.

Dla nauczyciela

Autor: Zespół autorski Contentplus.pl sp. z o.o.
Przedmiot: Informatyka

Temat: Ciąg Fibonacciego w języku Java

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

2) stosuje przy rozwiązywaniu problemów z różnych dziedzin algorytmy poznane w szkole podstawowej oraz algorytmy:

e) obliczania wartości elementów ciągu metodą iteracyjną i rekurencyjną, w tym wartości elementów ciągu Fibonacciego.

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Przeanalizujesz rekurencyjny algorytm generujący kolejne elementy ciągu Fibonacciego.
- Napiszesz programy wyznaczające elementy ciągu Fibonacciego w sposób rekurencyjny oraz iteracyjny.
- Rozwiążesz kilka zadań związanych z tematem e-materiału.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji).

Przebieg lekcji

Faza wstępna:

1. Nauczyciel wyświetla uczniom temat, wskazuje cele zajęć oraz ustala z uczestnikami zajęć kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel wyświetla na tablicy pytania zawarte w sekcji „Wprowadzenie":
   – co to jest ciąg Fibonacciego?
   – czym jest rekurencja?
   – do czego służy instrukcja warunkowa?
   Chętni uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. **Praca z tekstem**. Uczniowie analizują przykłady z sekcji „Przeczytaj" i powtarzają zaprezentowane rozwiązania na swoim komputerze. Uczniowie w parach rozwiązują problem 1, a następnie porównują swoje rozwiązanie z przedstawionym w prezentacji. W kolejnym kroku dzielą się swoimi spostrzeżeniami na forum klasy.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Gra edukacyjna". Uczniowie odpowiadają na pytania zaprezentowane w grze edukacyjnej.

3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1 i 2 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy w zakresie programowania w języku Java.

Praca domowa:

1. Uczniowie zapisują krótką notatkę podsumowującą różnice między iteracyjnym a rekurencyjnym algorytmem obliczenia wyrazu ciągu Fibonacciego o indeksie n

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Java SE 8 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Eclipse 4.4 (lub nowszej wersji).

Wskazówki metodyczne:

- Nauczyciel może wykorzystać multimedium w sekcji „Gra edukacyjna" do pracy przed lekcją. Uczniowie zapoznają się z jego treścią i przygotowują do pracy na zajęciach w ten sposób, żeby móc samodzielnie rozwiązać zadania dołączone do e-materiału „Ciąg Fibonacciego w języku Java".