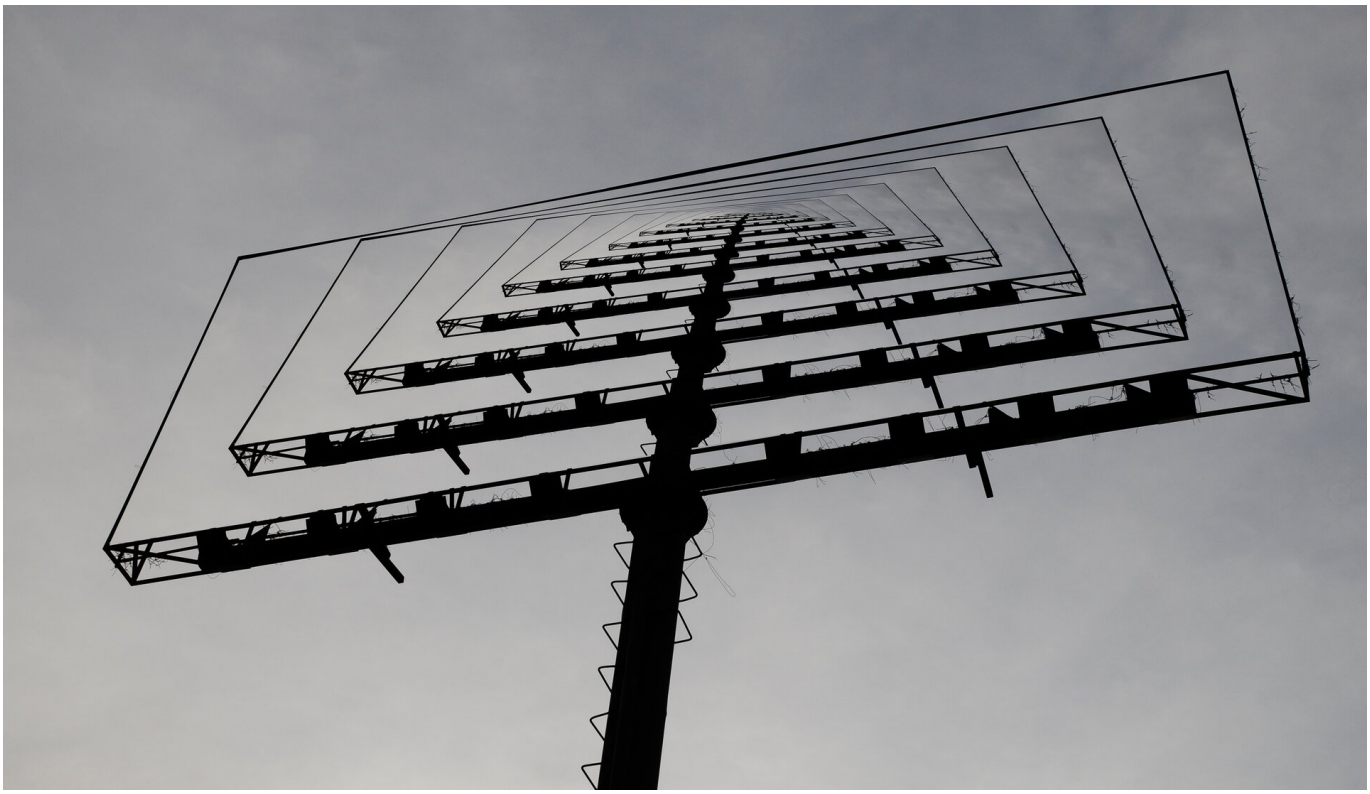


Rekurencja

- Wprowadzenie
- Przeczytaj
- Aplet
- Prezentacja multimedialna
- Dla nauczyciela

Rekurencja

Źródło: Pixabay, domena publiczna.

W internecie możesz znaleźć wiele prób wyjaśnienia, na czym polega rekurencja. Klasycznym już przykładem wyjaśnienia postępowania rekurencyjnego jest to, które zaproponował Andriej P. Jerszow.

Otóż określił on czynność **jedz kaszkę** jako: **jeśli talerz jest pusty, to zakończ jedzenie, w przeciwnym wypadku zjedz łyżkę kaszki i wróć do początku**.

Z definicji wynika, że dopóki na talerzu jest kaszka, należy zjeść jej jedną łyżkę i ponownie wykonać czynność **jedz kaszkę**. Jeżeli na talerzu brak kaszki, to czynność jest przerywana.

Z samą rekurencją możesz się spotkać w wielu – często dość odległych – dziedzinach, np. w sztuce, architekturze, optyce czy matematyce. W informatyce używamy jej między innymi w algorytmach generowania ciągów liczb, wyszukiwania wartości, sortujących zbiory czy generowania fraktali. W tym e-materiale omówimy definicję rekurencji oraz jej przykładowe zastosowania w informatyce.

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.

Zastosowanie rekurencji w poszczególnych językach programowania przedstawiamy w e-materiałach:

- Rekurencja w języku C++,
- Rekurencja w języku Java,
- Rekurencja w języku Python.

Więcej zadań? Sięgnij do: Rekurencja – ćwiczenia, Rekurencja w zadaniach.

W tym e-materiale odwołujemy się również do informacji dotyczących ciągu Fibonacciego. Omawiamy go w następujących e-materiałach:

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

> **Twoje cele**
>
> - Wyjaśnisz, na czym polega rekurencja, a także podasz przykłady jej zastosowań.
> - Przedstawisz rekurencyjną realizację algorytmu Euklidesa, obliczania silni oraz generowania ciągu Fibonacciego.
> - Wyjaśnisz, czym są liczby względnie pierwsze.

- Wskażesz ograniczenia, jakie wiążą się z wykorzystaniem rekurencji w programowaniu.
- Wymienisz przykłady rekurencji w sztuce.

Przeczytaj

Wiesz już, jak podchodzić do rozwiązywania stawianych przed tobą problemów programistycznych. Schemat wygląda tak, że zwykle najpierw próbujemy skorzystać z gotowego rozwiązania, a jeśli to nie zadziała, modyfikujemy i rozszerzamy rozwiązania, które znamy. Ostatecznie sięgamy po metodę łączenia rozwiązań częściowych. Próbujemy zatem uprościć proces. Są jednak problemy, które możemy rozwiązać, rozbijając je na takie podproblemy, które są podobne do całości. Rozwiązywanie mniejszych podproblemów ma zaś doprowadzić do rozwiązania problemu głównego. Taką metodę nazywamy metodą **rekurencyjną**.

Definicja rekurencji

Z rekurencją w informatyce spotykamy się w sytuacji, gdy korzystamy z rozwiązań tego samego problemu, ale dla mniejszej wartości lub liczby danych. W takim wypadku często stosujemy algorytm rekurencyjny, czyli odwołujący się do siebie. Algorytm kończy się, gdy potrafimy podać odpowiedź bez korzystania z rozwiązań dla mniejszych danych.

Przejdźmy do matematycznej definicji ciągu zdefiniowanego rekurencyjnie. Najpierw zapisujemy wyraz ciągu o indeksie 0 (lub kilka początkowych jego wyrazów). Następnie podajemy wzór na wyraz o indeksie n wyrażony za pomocą wyrazów poprzednich.

Wzór rekurencyjny uzależnia więc wartość dowolnego (ogólnego) wyrazu tego ciągu od wartości poprzedzających go wyrazów.

> **Przykład 1**
>
> Zapiszmy za pomocą pseudokodu algorytm jedzenia kaszki, który przedstawiliśmy w sekcji „Wprowadzenie”.
>
> ```
> Jedz kaszkę
>     jeśli talerz jest pusty
>         koniec jedzenia
>     w przeciwnym razie
>         zjedz łyżkę kaszki
>         Jedz kaszkę
> ```
>
> By lepiej zobrazować działanie algorytmu, zapiszmy go tak:
>
> ```
> Jedz kaszkę
> ```

```
    jeśli talerz jest pusty
        koniec jedzenia
    w przeciwnym razie
        zjedz łyżkę kaszki
        Jedz kaszkę
            jeśli talerz jest pusty
                koniec jedzenia
            w przeciwnym razie
                zjedz łyżkę kaszki
                Jedz kaszkę
                    ...
```

Zauważyć można, że sekwencja się powtarza – jedzenie kaszki (w programie byłaby to funkcja) powtarza się w ramach jedzenia kaszki (czyli funkcja wywołuje funkcję).

Przykład 2

Zapoznajmy się z przykładem matematycznym.

Jak wyznaczyć wyraz ciągu o indeksie `n` z ciągu zdefiniowanego rekurencyjnie?

Oto definicja pewnego ciągu określonego rekurencyjnie. Wyraz ciągu o indeksie 0 jest podany, np.:

$$a_n = \begin{cases} 2 & dla \quad n = 0 \\ a_{n-1} + 3 & dla \quad n > 0 \end{cases}$$

W jaki sposób możemy wyznaczyć wyraz ciągu o indeksie `n`? Przeanalizuj algorytm zapisany za pomocą pseudokodu wykorzystujący funkcję rekurencyjną.

Specyfikacja problemu:

Dane

- `n` – liczba całkowita; indeks wyrazu w ciągu, który chcemy obliczyć

Wynik

- obliczony wyraz ciągu o indeksie `n`

```
funkcja rekurencja(n):
    jeżeli n = 0:
        zwróć 2
    w przeciwnym razie:
```

```
5          zwróć rekurencja(n - 1) + 3
```

Największą zaletą funkcji rekurencyjnych jest prostota ich zapisu. Algorytm jest krótki i zrozumiały.

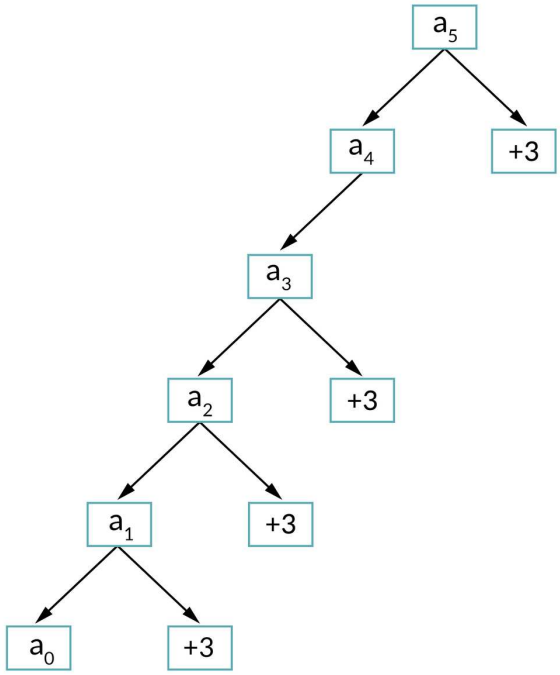
W przykładzie została opisana za pomocą pseudokodu funkcja rekurencyjna, która oblicza `n`-ty wyraz ciągu według podanego wzoru rekurencyjnego. Jej parametrem jest indeks wyrazu ciągu, który chcemy obliczyć.

Jeżeli argumentem funkcji będzie wartość 0, to zwróconą wartością będzie 2. Jeśli nie, ponownie wywoływana jest funkcja rekurencyjna, ale z inną wartością argumentu. W tym przypadku wartością argumentu jest indeks poprzedniego wyrazu ciągu, czyli `n - 1`.

Algorytm jest rekurencyjny wtedy, gdy do rozwiązania problemu wykorzystuje on sam siebie. Funkcja jest rekurencyjna wtedy, gdy wywołuje samą siebie. Kolejne wywołania takiej funkcji nazywamy **rekurencyjnym ciągiem wywołań**.

Rekurencyjny ciąg wywołań musi być skończony ten nie może być nieskończony, stąd w rekurencji musi być określony warunek kończący rekurencję.

Proces taki możemy pokazać za pomocą drzewa wywołań rekurencyjnych. Przedstawiony na ilustracji przykład jest drzewem wywołań rekurencyjnych dla początkowego wywołania funkcji `a(5)`. Takie wywołanie zwróci wartość 17.



Algorytm Euklidesa

Przypomnijmy: algorytm Euklidesa stosowany jest do wyznaczania największego wspólnego dzielnika (NWD) dwóch liczb naturalnych. NWD może być wykorzystany do skracania ułamków czy obliczania najmniejszej wspólnej wielokrotności (NWW). Więcej na temat tego algorytmu przeczytasz w e-materiale: Algorytm Euklidesa.

Specyfikacja problemu:

Dane

- `a` – liczba naturalna
- `b` – liczba naturalna

Wynik

- NWD liczb `a` i `b`

Algorytm Euklidesa z odejmowaniem

Wersja rekurencyjna prezentuje się następująco:

```
funkcja NWD(a, b)
    jeżeli a != b:
        jeżeli a > b
            zwróć NWD(a - b, b)
        w przeciwnym razie
            zwróć NWD(a, b - a)
    zwróć a
```

Funkcje są rekurencyjnie wywoływane do momentu, gdy wartości argumentów funkcji będą takie same. Wtedy zwracana jest wartość jednego z nich i to on jest największym wspólnym dzielnikiem liczb `a` oraz `b`.

Algorytm Euklidesa opiera się na prostej obserwacji: jeżeli liczba naturalna `c` dzieli liczby naturalne `a` i `b`, to dzieli również ich różnicę. Rzeczywiście jeśli `c = NWD(a, b)`, wówczas istnieją takie **względnie pierwsze** liczby naturalne `k` i `l`, dla których `a = c * k` oraz `b = c * l`. Niech ponadto `a > b`. Wówczas `a - b = c * k - c * l = c * (k - l)`, co oznacza, że liczba `a - b` również dzieli się przez `c`. To oznacza, że zarówno różnica liczb `b` oraz `a - b`, jak i różnica liczb `a` oraz `a - b` dzielą się przez `c`. Powtarzanie odejmowania doprowadzi do obliczenia `c`.

Algorytm Euklidesa z resztą z dzielenia

W tym wariancie algorytmu do znalezienia NWD – zamiast różnicy dwóch liczb – używa się reszty z ich dzielenia.

W podejściu rekurencyjnym dla `b` różnego od 0 przykładowa funkcja `NWD(a, b)` wywołuje samą siebie z argumentami `NWD(b, a mod b)` do momentu trafienia na przypadek podstawowy, kiedy `b` będzie równe 0. Pseudokod takiej funkcji wygląda następująco:

```
funkcja NWD(a, b):
    jeżeli a != 0:
        zwróć NWD(b mod a, a)
    w przeciwnym razie
        zwróć b
```

Funkcje są rekurencyjnie wywoływane do momentu, gdy wartość jednego argumentu funkcji jest równa 0. Wtedy zwracana jest wartość jednego z nich i to on jest największym wspólnym dzielnikiem liczb `a` oraz `b`.

Zastanówmy się, dlaczego możemy w ten sposób podejść do tego problemu.

Załóżmy, że chcemy obliczyć NWD dwóch liczb `a` oraz `b`. Są to liczby naturalne. Chcemy wyznaczyć liczbę `c`, która jest ich największym wspólnym dzielnikiem.

W naszym problemie `a ≤ b`. Podzielmy `b` przez `a`. Otrzymamy **iloraz** (`x` oraz **resztę** (`r`). Zapiszmy to.

$$b \div a = x \, reszta \, r$$

Zatem:

$$b = ax + r$$

natomiast:

$$0 \leq r < m$$

Jeśli `r = 0`, to `NWD(a, b) = a`. Oznacza to, że jedna z liczb dzieli drugą bez reszty, zatem mniejsza z liczb jest wspólnym dzielnikiem.

Jeśli natomiast `r ≠ 0`, możemy zapisać, że `r = b - xa`. Możemy zatem zauważyć, że każda liczba, która dzieli liczby `a` oraz `b`, dzieli również całe wyrażenie `b - xa`, zatem dzieli również `r`. Wynika z tego to, że NWD liczb `a` oraz `b` dzieli również resztę `r`.

Możemy to zapisać jako `NWD(a, b) = NWD(r, a)`, a ponieważ reszta z dzielenia jest wynikiem operacji modulo, to `NWD(a, b) = NWD(b mod a, a)`.

Problemy związane z rekurencją

Programy wykorzystujące rekurencję wymagają użycia dużej ilości pamięci. Dla każdego wywołania funkcji (w tym rekurencyjnego) program musi umieścić w pamięci adres powrotu, wartości zmiennych lokalnych, jak również wartości argumentów wywoływanej funkcji. Dane te są niezbędne do odtworzenia stanu przed wywołaniem, w momencie zakończenia danej funkcji. Może to spowodować przepełnienie pamięci. Informacje te zapisywane są w wyznaczonym miejscu pamięci, które zwiemy stosem.

Rozwiązywanie niektórych problemów metodą rekurencyjną może także znacząco zwiększyć złożoność czasową algorytmu. W takiej sytuacji wskazane jest użycie wersji iteracyjnej rozwiązania problemu.

Aby zobrazować problem przepełnienia stosu, posłużmy się przykładem. Uruchomienie funkcji `rekurencja(n)`, zdefiniowanej na początku tego e-materiału, spowoduje uruchomienie drzewa rekurencyjnego o wysokości n. Oznacza to, że aby obliczyć wartość funkcji `rekurencja(n)`, potrzebujemy na stosie obszaru o wielkości $n \cdot c$, gdzie c jest pewnym współczynnikiem stanowiącym średnią ilości pamięci zajmowanej na stosie przy każdym kolejnym wywołaniu funkcji.

> **Ważne!**
>
> Miejsce na stosie zajmowane przez funkcję rekurencyjną zależy od liczby wywołań rekurencyjnych i ilości danych przechowywanych na stosie przez każde wywołanie.
>
> W przypadku funkcji `rekurencja(n)`, która wywołuje samą siebie dwukrotnie z argumentami `(n - 1)` i `(n - 2)`, na stosie trzeba przechować dwa argumenty (typu `int`) oraz wartość zwracaną przez każde z dwóch wywołań. Ponieważ typ `int` zwykle zajmuje 4 bajty, to każde wywołanie funkcji `rek` zajmuje na stosie około 12 bajtów (2 x 4 bajty na argumenty i 4 bajty na wartość zwracaną).Dla przykładu, dla `n = 5` funkcja rekurencja wykona 15 wywołań rekurencyjnych, a więc zajmie na stosie około 180 bajtów (15 x 12 bajtów). Jednak w przypadku większych wartości `n` ilość danych na stosie będzie znacznie większa, co może prowadzić do przepełnienia stosu.

Niech c wynosi 16 bajtów. Wówczas przy zakładanej wielkości stosu maksymalna wartość n, dla której możemy wywołać funkcję `rekurencja(n)` bez błędu przepełnienia stosu, wynosi:

$$\frac{1\ MB}{16\ B} = \frac{1024 \cdot 1024\ B}{16\ B} = 65536$$

W rzeczywistości program może zakończyć się dużo wcześniej. Np. interpreter języka Python domyślnie ogranicza liczbę wywołań rekurencyjnych do 1000. W języku C++ nie ma takiego ograniczenia.

Złożoność obliczeniowa tej funkcji wynosi `O(n)`, ponieważ funkcja wywołuje samą siebie `n` razy, a każde wywołanie wykonuje stałą liczbę operacji (jedno dodawanie i jeden warunek).

Złożoność pamięciowa jest również `O(n)`, ponieważ dla każdego wywołania funkcji rezerwowany jest nowy stos o stałym rozmiarze (w tym przypadku 4 bajty) na wartości zwracane przez wywołania rekurencyjne oraz zmienne lokalne. Zatem całkowita ilość pamięci potrzebna na utworzenie wszystkich stosów wynosi `O(n * 4) = O(n)`.

Ciekawostka

Źródło: domena publiczna.

Efekt Droste'a swoją nazwę zawdzięcza opakowaniu kakao o tej samej nazwie. Na etykiecie produktu przedstawiona została kobieta trzymająca tacę z filiżanką i puszką kakao, na której w sposób rekurencyjny powtórzony został cały rysunek.

Podobny efekt zastosowano również na opakowaniach popularnego serka. Co ciekawe, jego nazwa w Polsce to fonetyczny zapis skróconej nazwy francuskiego oryginału (fr. *La vache qui rit*).

Innym przykładem rekurencji mogą być matrioszki, czyli rosyjskie drewniane lalki. Ich specyfika polega na tym, że wewnątrz największej lalki znajdują się identyczne, coraz mniejszych rozmiarów.

Problem 1

Wskaż przykłady rekurencji w sztuce.

Słownik

iteracja

technika programowania polegająca na powtarzaniu tych samych operacji w pętli określoną liczbę razy lub do momentu, aż zostanie spełniony zadany warunek

NWD

największy wspólny dzielnik dwóch liczb – największa liczba naturalna, która dzieli obie te liczby bez reszty

przepełnienie pamięci

błąd polegający na przekroczeniu rozmiaru zarezerwowanego miejsca dla programu w pamięci operacyjnej

rekurencja

technika programowania polegająca na odwoływaniu się procedur lub funkcji do samych siebie, aż do momentu spełnienia warunku podstawowego

stos

dynamiczna struktura danych, w której informacje pobierane są ze szczytu i na niego odkładane; struktura typu LIFO (*Last-In, First-Out* – ostatni na wejściu, pierwszy na wyjściu)

liczby względnie pierwsze

liczby całkowite, których największym wspólnym dzielnikiem jest liczba jeden

Aplet

Polecenie 1

Dany jest ciąg zdefiniowany rekurencyjnie:

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

Jest to ciąg Fibonacciego. W jaki sposób obliczyć jego wyraz o indeksie n?

Rozpisz drzewo wywołań rekurencyjnych dla podanego ciągu. Następnie porównaj je z tym zaprezentowanym w aplecie.

Specyfikacja problemu:

Dane

- n – liczba naturalna; indeks wyrazu w ciągu Fibonacciego

Szukane

- `wyraz` – liczba naturalna; obliczony wyraz ciągu Fibonacciego

```
funkcja fib(n):
    jeżeli n < 2:
        zwróć n
    zwróć fib(n - 1) + fib(n - 2)
```

Uruchom aplet prezentujący kolejne wywołania rekurencyjne funkcji. Przejdź przez poszczególne etapy wykonania procedury zarówno dla domyślnych, jak i własnych danych wejściowych. Jak zaprezentowane drzewo wywołań ma się do tego, które zostało przedstawione w sekcji „Przeczytaj”?

> **Ważne!**
>
> Wykorzystanie mechanizmu rekurencji do rozwiązania tego problemu jest nieefektywne. Funkcja będzie wielokrotnie obliczać te same wartości, z każdym wywołaniem `fib(n)`

licząc od nowa wartości `fib(n - 1)` i `fib(n - 2)`. Zwróć uwagę, ile razy dla wywołania `fib(5)` zostanie policzona wartość `fib(2)`. Funkcję zaprezentowano tylko w celu wyjaśnienia rekurencji. Nie należy jej stosować do wyznaczenia wyrazów ciągu Fibonacciego.

Zasób interaktywny dostępny pod adresem https://zpe.gov.pl/a/D1HpTu0N1

Polecenie 2

Dany jest następujący wzór zdefiniowany rekurencyjnie:

$$a_n = \begin{cases} 1 & dla \quad n = 0 \\ a_{n-1} * n & dla \quad n > 0 \end{cases}$$

Rozpisz drzewo wywołań rekurencyjnych ciągu dla $n = 4$.

Prezentacja multimedialna

Polecenie 1
Zapisz za pomocą pseudokodu algorytm obliczania silni liczby n. Wykorzystaj metodę rekurencyjną.

Specyfikacja problemu:

Dane:

- n – liczba naturalna, której silnię chcemy obliczyć

Wynik:

- liczba naturalna dodatnia; wartość silni liczby n $(n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n)$

Polecenie 2

Ciąg a_n dany jest rekurencyjnie:

$$a_n = \begin{cases} 4 & dla \quad n = 0 \\ 12 & dla \quad n = 1 \\ 3 * a_{n-2} + 7 & dla \quad n > 1 \end{cases}$$

Napisz za pomocą pseudokodu algorytm wyznaczający n-ty wyraz tego ciągu.

Specyfikacja problemu:

Dane:

- n – liczba naturalna dodatnia; indeks wyrazu w ciągu, który chcemy obliczyć

Wynik:

- liczba naturalna dodatnia; obliczony wyraz ciągu

Polecenie 3

Porównaj swoje rozwiązania z przedstawionymi w prezentacji.

Pokaż ćwiczenia:

Ćwiczenie 1

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Rekurencja

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy i rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

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

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

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.

3) wyróżnia w problemie podproblemy i charakteryzuje: metodę połowienia, stosuje podejście zachłanne i rekurencję;

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) zapisuje za pomocą listy kroków, schematu blokowego lub pseudokodu, i implementuje w wybranym języku programowania, algorytmy poznane na wcześniejszych etapach oraz algorytmy:

a) algorytm Euklidesa w wersji iteracyjnej i rekurencyjnej wraz z zastosowaniami,

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

b) rekurencję (do generowania ciągów liczb, potęgowania, sortowania liczb, generowania fraktali),

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):

- Wyjaśnisz, na czym polega rekurencja, a także podasz przykłady jej zastosowań.
- Przedstawisz rekurencyjną realizację algorytmu Euklidesa, obliczania silni oraz generowania ciągu Fibonacciego.
- Wyjaśnisz, czym są liczby względnie pierwsze.
- Wskażesz ograniczenia, jakie wiążą się z wykorzystaniem rekurencji w programowaniu.
- Wymienisz przykłady rekurencji w sztuce.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

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

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.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Rekurencja". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć. Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające prosi o ciche zapoznanie się z treścią w sekcji „Przeczytaj".
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Aplet". Uczniowie wspólnie analizują kolejne wywołania funkcji rekurencyjnej wyznaczającej n-ty wyraz ciągu Fibonacciego. Następnie powtarzają procedurę dla własnych danych wejściowych. W kolejnym kroku wykonują polecenie 2.
3. Uczniowie wykonują ćwiczenie z sekcji „Prezentacja multimedialna".

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat i cele lekcji zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji następuje omówienie ewentualnych problemów z rozwiązaniem ćwiczeń z sekcji „Prezentacja multimedialna".
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie rozwiązują polecenia 1 i 2 z sekcji „Prezentacja multimedialna". Następnie porównują swoje rozwiązanie z przedstawionym w prezentacji. W kolejnym kroku testują zapisany pseudokod dla przykładowych wartości n.

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj" można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.
- Uczniowie w ramach przygotowania do zajęć mogą poszukać przykładów obecności rekurencji w sztuce, naturze *etc.*
- Aplet można wykorzystać do wizualizacji działania rekurencji oraz stosu.