

Aufgabe 2

Beispiel 1: Sudoku in Python

Prompt:

Was macht dieses Programm?

```
board = [[0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 1, 9, 5, 0, 0, 0],
         [0, 9, 8, 0, 0, 0, 0, 6, 0],
         [8, 0, 0, 0, 6, 0, 0, 0, 3],
         [4, 0, 0, 0, 0, 3, 0, 0, 1],
         [7, 0, 0, 0, 2, 0, 0, 0, 6],
         [0, 6, 0, 0, 0, 0, 2, 8, 0],
         [0, 0, 0, 4, 1, 9, 0, 0, 5],
         [0, 0, 0, 0, 8, 0, 0, 7, 9]]
```

```
board2 = [[0, 2, 3, 4, 5, 6, 7, 8, 9],
          [4, 5, 6, 7, 8, 9, 1, 2, 3],
          [7, 8, 9, 1, 2, 3, 4, 5, 6],
          [2, 3, 4, 5, 6, 7, 8, 9, 1],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0], ]
```

```
def possible(x: int, y: int, n: int) -> bool:
    global board
    for i in range(9):
        if board[i][x] == n:
            return False
    for i in range(9):
        if board[y][i] == n:
            return False
    x0 = (x//3)*3
    y0 = (y//3)*3
    for i in range(3):
```

```
    for j in range(3):
        if board[y0 + i][x0 + j] == n:
            return False
    return True
```

```
def display():
    global board

    for i in board:
        index = board.index(i)

        print(board[index][:3], end=" ")
        print(board[index][3:6], end=" ")
        print(board[index][6:9])

        if index == 2 or index == 5:
            print("=====")
```

```
def solve():
    global board

    for y in range(9):
        for x in range(9):
            if board[y][x] == 0:
                for i in range(1, 10):
                    if possible(x, y, i):
                        board[y][x] = i
                        solve()
                        board[y][x] = 0
                return
    display()
    input("more?")
```

```
solve()
```

Antwort:

Dieses Programm ist eine Python-Implementierung eines Algorithmus zur Lösung eines 9x9-Sudoku-Rätsels mittels rekursiver Backtracking-Methode.

Es beginnt mit der Definition zweier Sudoku-Boards, board und board2, wobei board ein ungelöstes Rätsel und board2 ein gelöstes Rätsel ist.

Die Funktion `possible(x: int, y: int, n: int) -> bool` überprüft, ob es möglich ist, die Zahl `n` am Koordinatenpunkt `(x, y)` zu platzieren, ohne die Sudoku-Regeln zu verletzen. Dazu prüft die Funktion, ob `n` bereits in derselben Zeile, Spalte oder im 3x3-Teilquadrat vorhanden ist. Sie gibt `True` zurück, wenn `n` an dieser Stelle platziert werden kann, andernfalls `False`.

Die Funktion `display()` gibt das aktuelle Sudoku-Board visuell auf der Konsole aus.

Die Hauptfunktion `solve()` verwendet die Backtracking-Methode, um das Sudoku-Rätsel zu lösen. Es durchläuft das gesamte Board und versucht, für jede leere Zelle eine geeignete Zahl zu finden, indem es die Funktion `possible()` aufruft. Wenn eine passende Zahl gefunden wird, setzt es sie ein, ruft sich dann selbst rekursiv auf, um den Prozess für das nächste leere Feld zu wiederholen. Wenn keine passende Zahl gefunden wird, wird ein Backtracking durchgeführt, und die vorherige Zelle wird zurückgesetzt. Sobald das Rätsel vollständig gelöst ist, wird das Ergebnis angezeigt und der Benutzer zur Eingabe aufgefordert.

Es ist wichtig zu beachten, dass die Methode des reinen BacktrackingPossible(`x, y, n`): und `solve()` möglicherweise nicht für alle Arten von Sudokus funktionieren. In einigen Fällen kann es zu einer endlosen Schleife führen oder das Rätsel gar nicht lösen. Dennoch bietet das Programm eine allgemeine Idee davon, wie die Logik hinter einer Backtracking-Lösung für ein Sudoku-Rätsel aussehen könnte.

Prompt 2:

Board 2 ist kein vollständig korrekt gelöstes Rätsel.

Außerdem möchte ich wissen was die Idee hinter dem Input Statement sein könnte.

Antwort 2:

Danke für den Hinweis. Sie haben Recht, Board 2 ist tatsächlich nicht korrekt gelöst - es handelt sich um ein Board, bei dem jede Zeile, Spalte und jedes 3x3-Quadrat die Zahlen 1 bis 9 in beliebiger Reihenfolge enthält, aber mehrere mögliche Kombinationen zulässt, was zu einer ungültigen Lösung führt.

Was das Input-Statement in der `solve()`-Methode betrifft: Dieses Statement dient dazu, dem Benutzer die Möglichkeit zu geben, nach der Anzeige der gelösten Sudoku-Platte zu entscheiden, ob er weitere Lösungen sehen möchte. Normalerweise würde die `solve()`-Methode alle möglichen Lösungen des Rätsels finden und anzeigen. Da dies im vorliegenden Code nicht der Fall ist, könnte das Input-Statement eine Möglichkeit sein, den Benutzer zu fragen, ob weitere Lösungen gesucht werden

sollen, um die Rekursion fortzusetzen und andere mögliche Lösungen anzuzeigen. Es scheint jedoch, dass der Code nicht vollständig implementiert ist, da die rekursive Lösung nicht für verschiedene Lösungen eines Rätsels geeignet ist. Das Input-Statement könnte also ein Überbleibsel aus einer früheren Version des Codes sein, die nicht weiter implementiert wurde.

Dies ist eine Vermutung, da der eigentliche Zweck des Input-Statements nicht klar ist, da das Programm nicht korrekt implementiert ist.

Bewertung:

Die Funktionsweise korrekt erkannt, obwohl kein Variablen- oder Funktionsname einzeln eindeutig auf ein Sudoku schließen lässt. Auch die rekursive Methode das Spiel zu lösen wurde korrekt beschrieben. Fehler wurden mit der Beschreibung der Variable „board2“ und dem `input()` statement im Rumpf der `solve()` Methode gemacht.

`board2` wurde fälschlicherweise als korrekt gelöstes Feld interpretiert, obwohl es sich dabei in Wahrheit um ein Feld mit sehr vielen Lösungsmöglichkeiten handelt.

Der Aufruf der `input()` Methode dient dazu, dass Programm zu pausieren, bis der Benutzer einen Input in die Konsole eingibt und die Enter Taste drückt. Dadurch wird eine einzelne Ausgabe der Lösungen erreicht.

Aufgabe 2: Unschöne Fibonacci

Prompt:

Bitte verbessere den folgenden Java Code:

```
class Fibonacci {
    // Methode fibonacci
    // Berechnet die Fibonacci-Zahl des Arguments n rekursiv
    static long fibonacci(long n){
        if (n == 1 || n == 2)
            {return 1;}
        else
            {return fibonacci(n-1) + fibonacci(n-2);}
    }

    // Methode fibonacci iterativ
    // Berechnet die Fibonacci-Zahl des Arguments n iterativ
    static long fibonacciIterativ(long n){
        long ergebnis = 1;
        long vorletzter = 1;
        long letzter = 1;
        if (n == 1 || n == 2)
            {return 1;}
        else
        {
            for (int i=3; i<n+1; i++){
                {
                    ergebnis = vorletzter + letzter;
                    letzter = vorletzter;
                    vorletzter = ergebnis;
                }
            }
            return ergebnis;
        }
    }

    // Methode main
    // Aufrufen der beiden Fibonacci-Zahl-Methoden von 1 bis 40
    public static void main(String[] args){
        for (int i=1; i<=40; i++){
            System.out.println(i + " Fibonacci iterativ : " + fibonacciIterativ(i) + "
            Fibonacci rekursiv : " + fibonacci(i));
        }
    }
}
```

Anwort:

Der angegebene Java-Code enthält eine Reihe von Verbesserungsmöglichkeiten, sowohl in Bezug auf die Codierungsstile als auch auf die Leistungsoptimierung des rekursiven Algorithmus. Ich werde den Code kommentieren und verbessern:

```
class Fibonacci {  
  
    // Methode fibonacciRekursiv  
    // Berechnet die Fibonacci-Zahl des Arguments n rekursiv  
    // Hinweis: Dieser rekursive Ansatz ist sehr ineffizient für große n,  
    // da er viele Berechnungen mehrfach durchführt.  
    static long fibonacciRekursiv(long n) {  
        if (n <= 0) { // Hier sollten wir prüfen, ob n negativ oder 0 ist  
            return 0; // Und entsprechend 0 zurückgeben, da negative Indizes  
nicht definiert sind  
        } else if (n == 1 || n == 2) {  
            return 1;  
        } else {  
            return fibonacciRekursiv(n - 1) + fibonacciRekursiv(n - 2);  
        }  
    }  
  
    // Methode fibonacciIterativ  
    // Berechnet die Fibonacci-Zahl des Arguments n iterativ  
    static long fibonacciIterativ(long n) {  
        if (n <= 0) { // Auch hier sollten wir prüfen, ob n negativ oder 0 ist  
            return 0; // Und entsprechend 0 zurückgeben  
        } else if (n == 1 || n == 2) {  
            return 1;  
        } else {  
            long ergebnis = 1;  
            long vorletzter = 1;  
            long letzter = 1;  
            for (int i = 3; i <= n; i++) {  
                ergebnis = vorletzter + letzter;  
                vorletzter = letzter;  
                letzter = ergebnis;  
            }  
            return ergebnis;  
        }  
    }  
  
    // Methode main  
    // Aufrufen der beiden Fibonacci-Zahl-Methoden von 1 bis 40  
    public static void main(String[] args) {  
        for (int i = 1; i <= 40; i++) {  
            System.out.println(i + " Fibonacci iterativ: " +  
fibonacciIterativ(i));  
            // Die rekursive Methode wird nicht für große 'i' aufgerufen, da  
dies zu erheblichen Leistungsproblemen führt  
        }  
    }  
}
```

Wichtige Änderungen:

Vorlesung Programmieren | Selbststudium | 14.05.2024 | Matti Bos

- Die Bedingung `n <= 0` wurde der Methode `fiborekursiv` und `fibolterativ` hinzugefügt, um sicherzustellen, dass nur für positive `n` die Fibonacci-Zahl berechnet wird. Für nicht-positive `n`, wie 0 oder negative Werte, gibt die Methode 0 zurück.
- Die Kommentare wurden verbessert, um die Änderungen zu erklären und eine Warnung über die Ineffizienz der rekursiven Methode für große `n` hinzuzufügen.
- In der `main`-Methode wird jetzt nur die iterative Methode für die Berechnung aus Gründen der Leistungsoptimierung verwendet. Die rekursive Methode kann für kleine `n` separat getestet werden, um ihre Funktionalität zu überprüfen, sollte aber für große `n` wegen ihrer Ineffizienz vermieden werden.

Bewertung:

Die Programme sind beide funktionsfähig. ChatGPT hat dabei auch auf die deutlich langsamere Ausführungszeit der rekursiven Implementierung hingewiesen. In der Main Methode wird diese langsamere Funktion nicht mehr aufgerufen um Zeit zu sparen, im Sinne der Optimierung ist dies zwar sinnvoll, steht aber dem hier beabsichtigten Vergleich der beiden Varianten entgegen.

Insgesamt wurden die optischen Aspekte des Programmcodes verbessert, wie beispielsweise Leerzeichen zwischen den Operatoren innerhalb der „for“ Schleifen. Allerdings sind einige der Kommentare unnötig, da ihr Inhalt durch den Code selbst offensichtlich erkennbar ist und daher nicht explizit genannt werden müsste. Zum Beispiel ist ein Kommentar der die Main Methode als solche beschreibt völlig redundant.